RADC-TR-88-11, Vol VI (of eight) Part A
Interim Technical Report
[illegible]



AD-A198 285

NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986
Computer Architectures for Very Large Knowledge Bases

Syracuse University

Dr. P. Bruce Berra, et. al.

BEST AVAILABLE COPY

This effort was funded partially by the Laboratory Director's fund.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC ELECTE SEP 09 1988 S D E

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss AFB, NY 13441-5700

88 9 6 134

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-11, Volume VI (of eight), Part A has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

APPROVED: Raymond P. Urtz, Jr.

RAYMOND P. URTZ
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-11, Vol VI (of eight) Part A
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC) 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62702F (over) / PROJECT NO. 5581 / TASK NO 27 / WORK UNIT ACCESSION NO. 13
11. TITLE (Include Security Classification) NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986 Computer Architectures for Very Large Knowledge Bases	
12. PERSONAL AUTHOR(S) P. Bruce Berra, et. al.	
13a. TYPE OF REPORT Interim 13b. TIME COVERED FROM Jan 86 TO Dec 86 14. DATE OF REPORT (Year, Month, Day) June 1988 15. PAGE COUNT 232	
16. SUPPLEMENTARY NOTATION This effort was funded partially by the Laboratory Directors' Fund.	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
25	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Very Large Knowledge Bases, Artificial Intelligence, Computer Architectures, Real Time Processing, Pattern Matching, Parallel Computing. (KBS)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the second year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: a versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photo interpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system. The specific topic for this volume (Part A) is the development of architectures for very large knowledge bases, especially in light of real-time requests, parallelism, and the advent of optical computing.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Raymond A. Liuzzi	22b. TELEPHONE (Include Area Code) (315) 330-2643 22c. OFFICE SYMBOL RADC (COTD)

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

UNCLASSIFIED

Block 10 (Cont'd)

Program Element Number	Project Number	Task Number	Work Unit Number
62702F	4594	18	E2
61101F	LDFP	15	C4
61102F	2304	J5	01
33126F	2155	02	10

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC COPY INSPECTED

A.1 Computer Architectures for Very Large Knowledge Bases

Report submitted by: P. Bruce Berra
Electrical and Computer Engineering
Syracuse University
Syracuse, New York 13244-1240

Table of Contents

		Page
A.1	Project Goals and Status	1
A.2	Introduction	2
A.3	Partial Match Retrieval	3
A.4	Surrogate Files	5
A.5	Associative Memory	8
A.6	Back End System	11
A.7	Relational Operations on the Surrogate File	14
A.8	Optics	15
A.9	Current and Future Plans	16
A.10	References	20

Appendix A-1	Computer Architecture for a Surrogate File to a Very Large Knowledge Base	A-1-1
Appendix A-2	Optical Techniques in Data and Knowledge Base Machines	A-2-1
Appendix A-3	An Initial Design of a Very Large Knowledge Base Architecture	A-3-1
Appendix A-4	An Analysis of Surrogate File Structures for Very Large Knowledge Bases	A-4-1

A.1 Project goals and Status

The long term goal of this project is to develop innovative computer architectures that efficiently manage very large knowledge bases (VLKB) in a real time environment. There is a current need for expert systems with large and very large knowledge bases. With these systems comes the problem of the efficient management of the knowledge base. Database management system technology can help with smaller knowledge bases but when real time requirements and a very large knowledge base are involved one must consider new hardware approaches. The context of our knowledge base management research is that of logic programming. That is, the inferencing mechanism is written in a logic programming language with the rules and facts as Horn clauses.

Our current investigations have focused on three related areas: 1) the development of techniques for accessing the extensinal database (EDB) of facts in minimum time, 2) the development of parallel computer architectures that can further speed up EDB processing and 3) optical processing of the EDB. When one seeks to satisfy subgoals in a query, access to the EDB can take place on any subset of arguments that exist in the clause type under consideration. The problem then becomes one of partial match retrieval with some form of indexing over all argument positions. In order to reduce the amount of index data we have investigated the use of surrogate files that are composed of superimposed code words, concatenated code words, transformed inverted lists or combinations of all three. These are transformations of EDB key values into binary representations. While a code word exists for every fact in the EDB the total code word file is on the order of 20% of the size of the EDB of facts. This is compared with sizes of upwards of 100% for other methods of indexing.

With regard to the development of parallel computer architectures we have postulated a back end system that consists of a request processor, many surrogate file processors, disks for the storage of the surrogate file and the EDB, and an extensional database manager. We are currently invetigating several approaches to the surrogate file processor from associative processors to streaming processors.

In addition to considering an electronic solution to the problem we have also begun to consider an optical solution. One can approach it from two points of view. The first is to modify optical storage media in order to greatly increase the amount of data (300 M Bytes/sec) being sent to the digital computer and the second is to introduce an optical processor in between the modified optical disk and the electronic computer.

During the next year we will concentrate on the design of the back end system and the optical approach. Also, we will devote considerable effort to the interface between logic programming and relational data base management. This is important since these two technologies must be interfaced in an efficient way in order to optimize the performance of the entire system.

In subsequent years we will demonstrate a back end system that contains special hardware for the management of the EDB. This back end system will have been integrated with a front end logic programming system under development by Dr. Ken Bowen at Syracuse University. Following that we plan to develop a prototype system that is designed to address the more global issues involved in the management of very large knowledge bases in a real time environment.

A.2 INTRODUCTION

The current state of the art in knowledge based expert systems is such that the intensional database (IDB) of rules and the extensional database of

facts (EDB) are small and main memory resident. However, there is a current need for expert systems with large and very large knowledge bases. With these systems comes the problem of the efficient mamagement of the knowledge base. Data base management system technology can help with smaller knowledge bases but when real time requirements and a very large knowledge base are involved one must consider innovative hardware approaches. Thus, the long term goal of this research project is to develop innovative computer architectures that efficiently manage very large knowledge bases in a real time environment.

There are many ways to represent the knowledge in an expert system. We have chosen a logic programming framework because of its strong mathematical foundation, its commonality with relational data base management, prior and current Prolog and MetaProlog work at Syracuse University and the potential for making significant improvements in the performance of logic programs through the exploitation of search parallelism.

In this report we begin with a description of the partial match retrieval problem that results when one seeks to retrieve facts from the EDB. We discuss our aproach to the improvement of the performance of the partial match retrieval problem, which includes the use of a surrogate file. We then discuss various ways of structuring the surrogate file such as superimposed code words (SCW), concatenated code words (CCW) and trans-formed inverted lists (TIL). We discuss the comparative evaluation of the various methods for structuring the surrogate file and special computer architectures for the processing of the surrogate file. We also discuss an optical approach that we have been inversigating. Finally, we present current for and future plans.

A.3 PARTIAL MATCH RETRIEVAL

In our initial investigations we have taken the approach that the

intensional data base (IDB) of rules is separate from the extensional data base (EDB) of facts. We assume that the inferencing engine goes about the process of solving logic programming problems through processing of the rules and making calls to the EDB to obtain facts that are needed in the process. As previously mentioned we are concerned with a very large knowledge base (VLKB) of facts and high access time requirements. In order to illustrate that the problem at this level becomes one of partial match retrieval consider the following fact type.

Flight (Origin, Destination, Aircraft Type, Departure Time)

One instance of this fact type can be the following:

Flight (Vernon, Verona, B 787, 2:45 AM)

This is interpreted as the fact that a flight originates from Vernon at 2:45 AM, proceeds to Verona and the aircraft type is a B 787. In the process of solving a logic program the query to the EDB could translate into any of the following queries:

Where do flights originate at 2:45?

What kind of aircraft are used between Vernon and Verona?

Are there any flights of B 787 Aircraft at 2:45 AM between Vernon and Verona?

and many others.

The point here is that access may be required on any subset of the fact type arguments. The queries above represent access on one, two and all argument positions. The problem becomes a special case of the partial match retrieval problem. It is a special case because in general, we will know which argument positions we are interested in.

The partial match retrieval problem can be solved by first indexing on fact type and then creating an index on each argument position within a fact

type. While this approach effectively solves the problem for small EDB's it is ineffective for a VLKB. Since we are indexing on each argument position the index data can be as large as or larger than the facts themselves. In a small EDB with just a few megabytes of fact data doubling the size of the data base is not a severe price to pay for retrieval performance. However, if one has several gigabytes of fact data, doubling the total amount of data by creating indexes on each argument position does not represent a viable solution to the problem. This is true not only from the storage point of view but also from the accessing point of view since one must manage much more data. Thus, other methods must be found, that give improved performance when dealing with gigabytes of data.

A.4 SURROGATE FILES

Over the past year we have made detailed comparative evaluations of techniques that reduce the size of the index data and offer improved retrieval performance. The principal techniques that we have evaluated include superimposed code words (SCW), concatenated code words (CCW), combinations of SCW and CCW and transformed inverted lists (TIL). We will use superimposed code words to illustrate the ideas.

In our previous example suppose we use a hashing function to transform each of the arguments to a fixed binary representation (say 32 bits). That is,

H(Vernon)	100...1
H(Verona)	110...0
H(B 787)	110...1
H(2:45 AM)	110...1
Logical OR result	110...1 00...1

We can logically OR the individual binary code words to form a

superimposed code word. We would then attach a unique identifier to the code word. This unique identifier would be the same one that is used to identify the fact in its complete form. Thus, there would be a surrogate file of SCW's with one entry per fact and this file would be used to improve retrieval performance.

As illustrated in Figure A.1 if one wanted to retrieve a fact based upon one or more arguments, one would pass each of the arguments through the same hashing function, and OR their fixed length binary representation together in order to generate the query code word (QCW). One would then compare the QCW with all of the SCW's for that fact type seeking matches on the ones of the QCW. All of the SCW's that had ones in the same positions as the QCW would be possible qualifiers. The id's of the possible qualifiers would then be used to retrieve the pages in question. The input key values would then be compared with the respective key values of the retrieved facts in order to determine possible matches. If the desired fact(s) exists within the database it will be found. However, a number of pages may be retrieved that do not contain desired facts, called false drops. There is an inverse relationship between the number of bits used in the SCW and the number of false drops.

In the use of concatenated code words we concatenate the binary representation along with the unique identifier to form the CCW. Thus a fact becomes a tuple that adheres to the assumptions of the relational database model. The use of CCW's is one extreme while the method of SCW's is the other. The concatenation method almost insures (subject to the quality of the hashing function in avoiding collisions) that the unique identifier(s) that is obtained is indeed the one(s) that is desired. However, it does so at the expense of a longer word length. The method of SCW's reduces this long word length but does so at the expense of increasing the number of pages retrieved that do not

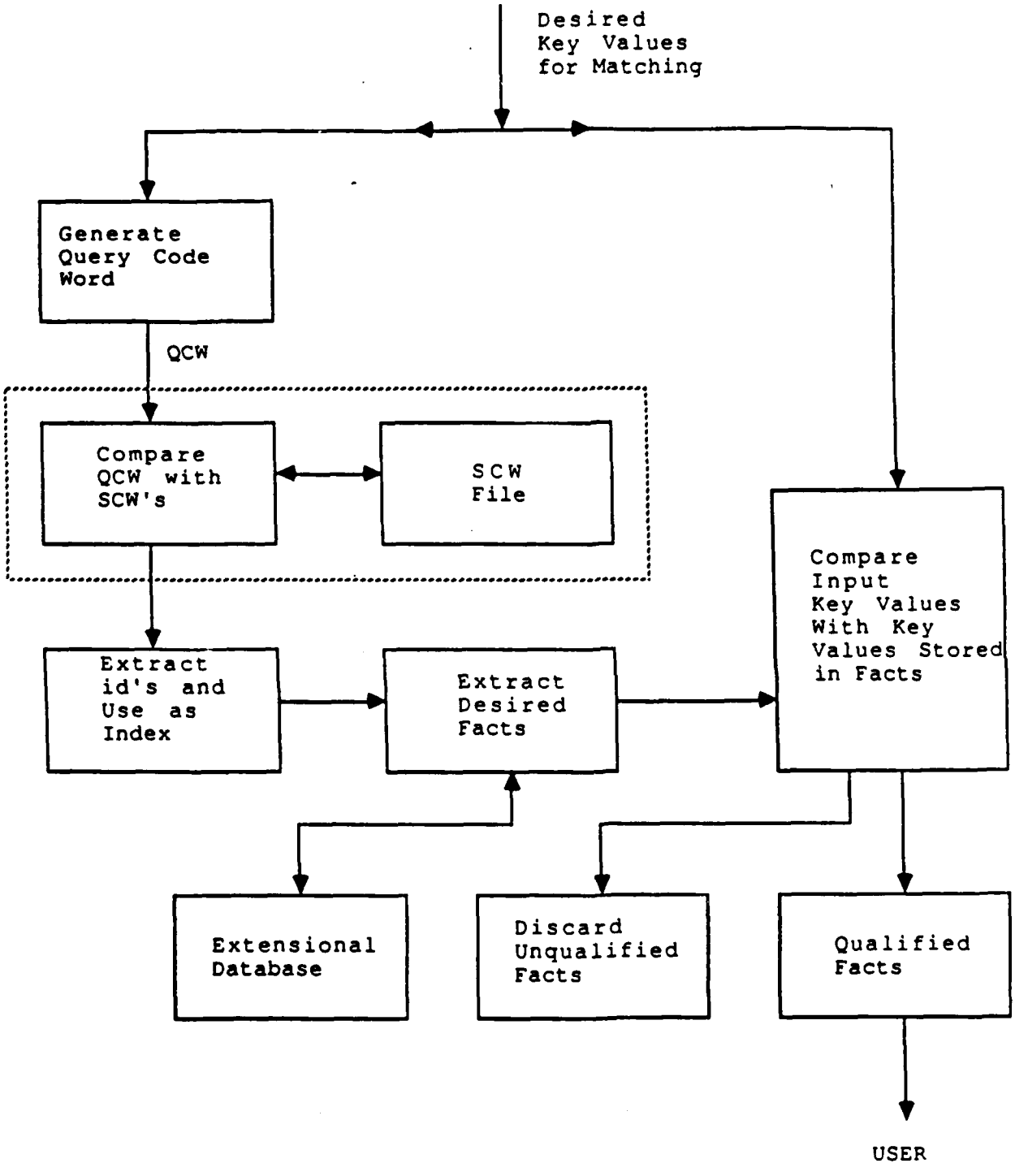


Figure A.1 Partial Match Retrieval Using Superimposed Code Words (SCW).

contain desired facts. Thus, there is a basic trade-off between these two methods.

Another approach that we have investigated is the use of transformed inverted lists. In this method we hash the original argument values for each position and keep a list with pointers to the original facts. If one is searching for a fact based on a single argument this method has definite advantages since less index data needs to be processed. However, when a number of arguments are involved additional processing will be required since lists will have to be intersected. Also, updating the list of facts will be more difficult than with SCW and CCW.

We have developed equations for the amount of storage space needed for all of the methods discussed above. In addition, we have developed detailed timing equations that determine the time required to retrieve the necessary part of the surrogate file, process it, retrieve the necessary facts, process them and return the results to the inferencing mechanism. We have used these equations as part of a simulation whereby we have obtained data that can be used for comparative purposes. Essentially, depending upon the assumptions that one makes about the application under consideration each of the methods have their strong and weak points (Berra, P.B., et al 1987). It should be noted that the comparisons have been made on the basis of sequential processing. This point is extremely important since the final solution lies in the match between the algorithm and the special computer architecture design. That is, an algorithm that words well on a sequential processor may not work well on a parallel processor and vice versa. As pointed out in the next section there seems to be advantages in the CCW technique but no final decision has been made as yet.

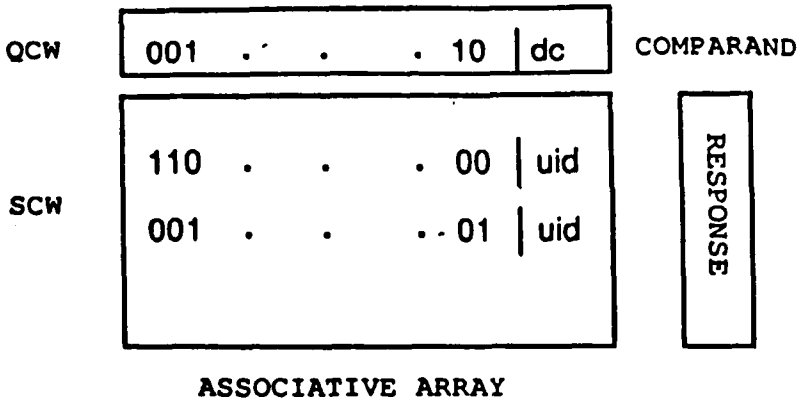
A.5 ASSOCIATIVE MEMORY

In this section we will consider the processing of the surrogate file by an associative memory when the file is constructed using SCW, CCW or TIL. Recall that under our previous assumptions all partial match retrievals will be performed using the surrogate file. Thus, retrieval of the facts will take only two disk accesses given the unique identifier using extensible hashing (Fagen, et al 1979).

Associative memories have the capability of performing a wide variety of operations on array resident data. These operations include exact match, maximum, minimum, etc. as well as Boolean operations. These devices are well known for performing these operations using content addressing and parallelism. However, these devices are generally costly, rigid in data format and Input/Output bound (Berra et al 1979). Nevertheless, when the associative memory is used to process the surrogate file one of these disadvantages goes away and another is mitagated. That is, the format of the surrogate file is highly regular and maps very well into the associative memory; and the surrogate file is a compact representation of the data being considered and less I/O is required.

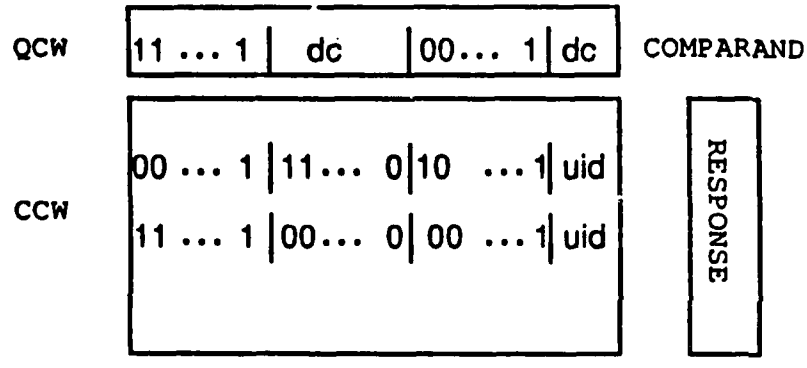
In the case of SCW, research by Ahuja and Roberts (1980) indicates that the mapping between the SCW file and the Associative Memory is very good. The processing of the array resident data takes place by performing an exact match search using the QCW as the search arguement as shown in Figure 9.1.2a. The unique identifiers for all of the SCW's that matched the QCW on the basis of ones would be sent to the fact manager for retrieval and further processing of the facts.

The processing of a CCW file would be similar but with some variation as shown in Figure A.2b. The QCW would have don't cares in those positions where



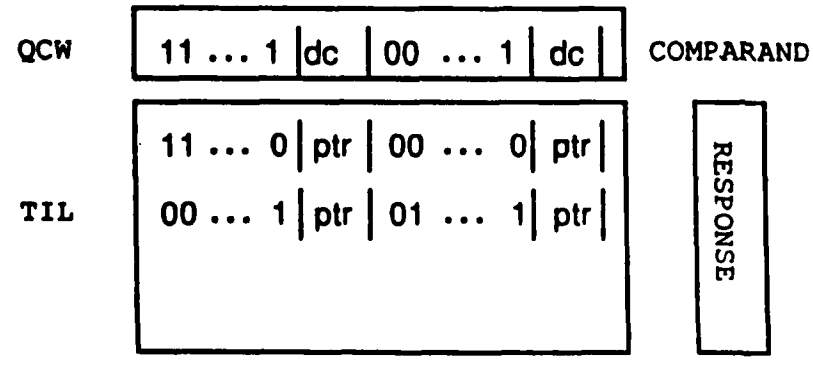
Search on the basis of ones

Figure A.2a SCW Mapped into Associative Memory



Search on the basis of ones and zeros

Figure A.2b CCW Mapped into Associative Memory



Search on the basis of ones and zeros

Figure A.2c TIL Mapped into Associative Memory

values are unknown and a known bit pattern in the positions of interest. An exact match search would be performed on the ones and zeros of the fields of interest, ANDed together and the selected unique identifiers sent on for further processing.

In order to process TIL files with an associative memory, as shown in Figure A.2c, one would need to form subfiles of unique identifiers for each value in each argument position since these values will not be unique. That is, each distinct value will have a pointer attached to it that points to the list of unique identifiers of the facts that contain that value. The processing of the TIL file will require less transfer of data to the associative memory for searches with few arguments but will require intersecting the lists of unique identifiers prior to passing the results on. However, if many arguments are given in the request, considerable processing will be required.

A.6 BACK END SYSTEM

Shown in Figure A.3 is a back end system for the management of a very large extensional data base of facts. The back end system will also manage many intensional data bases but those are not shown on the diagram. We assume that there are many gigabytes of fact data stored on the EDB disks. Likewise there are several gigabytes of surrogate file data stored on the SF disks. It should be pointed out that the SF and EDB are likely to reside on the same disk system. Since we have assumed the relational model we will store the facts by relation and then by tuple unique identifier within relations. As previously mentioned we will access only by relation name and then by tuple identifier, so extensible hashing or some such technique that minimizes disk accesses can be used. The surrogate file will really consist of many subfiles one for each relation. Each of the subfiles will be distributed uniformly over all of the SF disks in order

to insure maximum parallelism in disk accessing. As an example, suppose that the surrogate file is constructed with concatenated code words and the user is interested in retrieving fact data given some subset of values from a particular relation. The query code word would be constructed in the Request Processor using the proper hashing function and taking into consideration the positions of the values within the relation. The QCW would then be broadcast to all of the surrogate file processors (SFP) to be used as a search arguement. As discussed in the previous section, one could think of the SFP as an associative memory with the QCW as the search arguement. However, the full capabilities of the associative memory would not be needed for the operations discussed here. There is no point in providing much more capability in the SFP than is required to process the SF data at disk rates. Thus, the SFP would be rather simple since all it really needs to do is compare the QCW with each CCW at disk rates and strip off the unique identifiers. As soon as any unique identifiers are found by the SFP's they can be sent to the collector and quickly passed on to the Extensional Data Base Manager (EDBM) for processing thus providing some level pipelining. The EDBM will retrieve the facts, compare them with the search criteria to insure that a collision has not occurred, put them in blocks and send the blocks to the logic programming engine.

One would follow a similar process if the SF was constructed with SCW. The main difference is that extra processing would be required by the EDBM because of the false drops. Other work regarding processors for SCW's can be found in Lee (1986) and Colomb and Jagasooriah (1986).

If the SF is composed of TIL's the processing will be slightly different and an additional hardware unit will have to be added to the system shown in Figure A.3. Each of the lists would be distributed uniformly over the SF

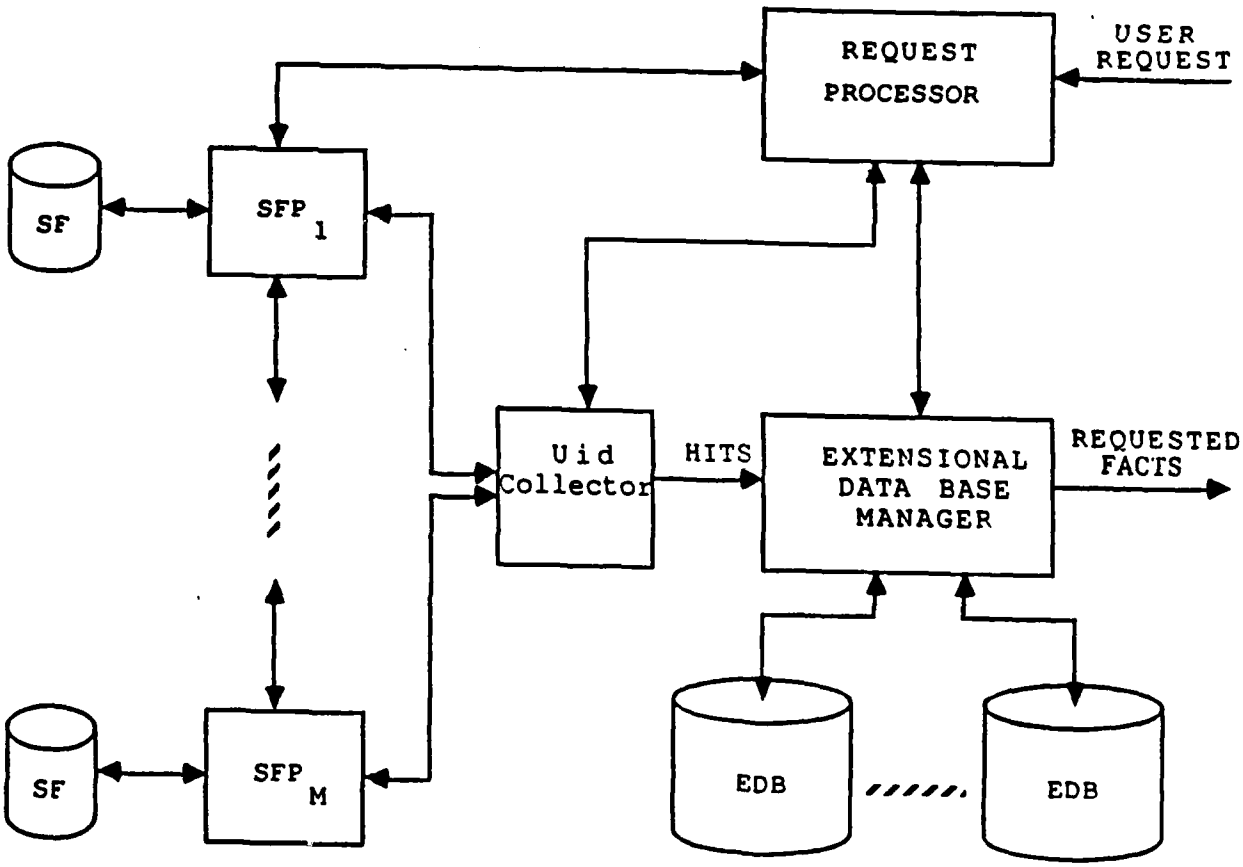


Figure A.3 Back End System for Fact Management

disks. As an example assume that a user's request requires access to only two lists. The first list would be input to the SFP's and a simple comparison made for an exact match. Note that the SFP is just a simple comparator in this case. The unique identifiers would be stripped off and sent to the collector. The second list would be processed in a similar way. However, now the two lists will have to be intersected. Thus, we will need an additional block between the collector and the EDBM called the intersector.

In this discussion we have tried to provide an idea of how such a system might work in retrieving facts from a very large extensional data base. Obviously, many more issues need to be addressed such as update, integrity and collisions.

A.7 RELATIONAL OPERATIONS ON THE SURROGATE FILE

A serendipitous benefit obtained from using the surrogate file approach is that some relational operations can be performed on the surrogate file rather than on the relations themselves. This offers considerable potential savings in time to carry out these relational operations and thus contributes an additional measure of performance to the system.

Consider the case in which the surrogate file is constructed with CCW's. To perform a selection on a particular field one just uses the surrogate file processor to do a partial match on that field. Extending this slightly it is clear that the SFP is just a selection mechanism. With a small amount of extra hardware the SFP should be called the selector.

Consider the performance of a join using the hash join algorithm. Since the surrogate file already consists of hashed values all that needs to be done is to assign the portion of the code word that represents the join variable and the unique identifier for each relation to buckets. Then based on matching

within each bucket, which can be done in parallel, pairs of unique identifiers can be sent to the EDBP for final verification. This procedure would work for the equijoin join but an order preserving hashing function (Garg and Gotlieb 1986) would be needed for performing other joins.

It appears that there would be little advantage to doing projections on the surrogate file. Thus, the EDBP would have to have the tuples stored in such a way that projections could be done directly on the tuples of the relation under consideration. This could be acomplished by treating each relation independently with its own extendible hashing system. Storing the relation this way would also aid in the performance of other operations that pertain to all or a large number of tuples in the relation.

Finally, range queries could be performed on the CCW file given a suitable order preserving hashing function; albiet with some additional processing such as sorting.

In the case of SCW it would be difficult to perform most relational operations on the surrogate file. In the case of TIL files selections and hash joins could be performed but projections would have to be performed on the tuples themselves. Range queries could also be performed given a suitable order preserving hashing function and some additional processing.

A.8 OPTICS

Another technology that holds promise for the management of very large knowledge bases is optics. The transfer rate from optical disks can potentially be increased to more than two orders of magnitude (300 M Bytes/sec) greater than the transfer rate from magnetic media. If this level of performance could be realized then a great number of I/O bound problems could now be solved. However, many machines have been designed to accept data at magnetic disk

transfer rates. Thus, this leads to a problem in the acceptance of the data. One possible solution is the develop an optical knowledge base machine that does processing on the stream of data coming from the optical disks. After the processing has been performed then the data rate to the electronic computer should be low enough for it to accept without modification. That is, the data stream would look like it was coming from a magnetic disk but would be much richer.

During this past year we have been investigating optical processing with a view toward the problems discussed above (Berra, et al 1986).

A.9 *CURRENT AND FUTURE PLANS*

During this reporting period our investigations have focused on three areas of research. The first is the development of mathematical models in order to evaluate the relative performance of superimposed code words, concatenated code words and transformed inverted lists as possible condidates for the surrogate file. We have developed equations that allow us to calculate the size of the surrogate file when constructed using CCW, SCW or TIL. We have also developed detailed timing equations that have been used to determine the amount of time it takes to respond to a user's query. These equations take into consideration the retrieval and processing of the surrogate file, the retrieval of the facts from the extensional data base and the processing of the facts prior to sending them to the inferencing mechanism. With these equations we have made comparative analysis of the three techniques under a wide variety of conditions (Berra, et al 1987).

The second main area of research is the development of a specialized computer architecture that will process the surrogate file in an optimal way. This architecture must be such that the match between the algorithm, whether it

is CCW, SCW or TIL, and the hardware be optional in some sense. The approach that we are considering is given in Figure A.3.

The third area of research concerns the use of modified optical disks that allow for the bandwidth between secondary storage and main memory to increase by at least two orders of magnitude. As part of this research we are investigating optical knowledge base machines that could be used to process the data in its optical form prior to being sent to the electronic computer.

During the past year a joint member of the Syracuse University teams interfaced prolog with a relational data base management system (Hughes 1986). This work is important in that it forms the bridge between our work which is aimed at dealing with very large knowledge bases and the Prolog/Meta Prolog work by Dr. Bowen which is aimed at solving problems that require very large knowledge bases.

During the next year (1987) we will wrap up our comparative evaluation of SCW, CCW, and TIL as algorithms for constructing surrogate files. We will then seek an optimal match between the algorithm and a special processor that we intend to build. The basic structure is as shown in Figure A.3. We will concentrate on the Surrogate File Processor design and by year end we expect to have the design completed so that we can move into prototype fabrication. We will then begin to assemble the necessary hardware, software and knowledge for the knowledge base back end system. We will have to pay special attention to those functions that are required for a complete system. These functions include creating the EDB, back up, update, integrity, security and others.

We will also concentrate on the interface between logic programming and relational data base management. While we have made some advances in this area there are many other issues to deal with. Beyond all of the basic interfacing

issues there are questions concerning how the extensional data base processor can optimally utilize advanced information from the inferencing mechanisms and vice versa. We plan to investigate this specific issue and others during the next year.

We will also continue with our optics investigations placing emphasis on the movement and processing of large amounts of data.

In our long term research, if we are to effectively handle VLKB's, we must deal with the storage of the knowledge in disk form. Since the bandwidth (3-5 M Bytes/sec) of magnetic disks is not expected to increase much during the time period of this research we must consider the distribution of the EDB over many disks. This will allow for simultaneous parallel read out from the disks, thus effectively increasing the bandwidth. Another way to increase the effective bandwidth of the disks is to compact/decompact the data being sent to and from the disks. Estimates are that compaction can effectively increase the bandwidth by a factor of two. Thus, with the minimization of the amount of extra data through the use of surrogate files and the methods described above we hope to be able to deal effectively with the magnetic disk bandwidth problem. Finally, we expect a positive impact based upon our optics research.

In the time period 1988 through 1989 we plan to prototype an advanced architecture as illustrated in Figure A.4. We will use the techniques described above for aiding rapid EDB retrieval but many other problems will arise regarding distributing the EDB, coordination and communication among the processors, the numbers of processors of various types needed for effective operation of the system, the establishment of a VLKB and the interfacing to the inferencing mechanism. Our long term goal then is to be able to demonstrate such a system by the end of the project.

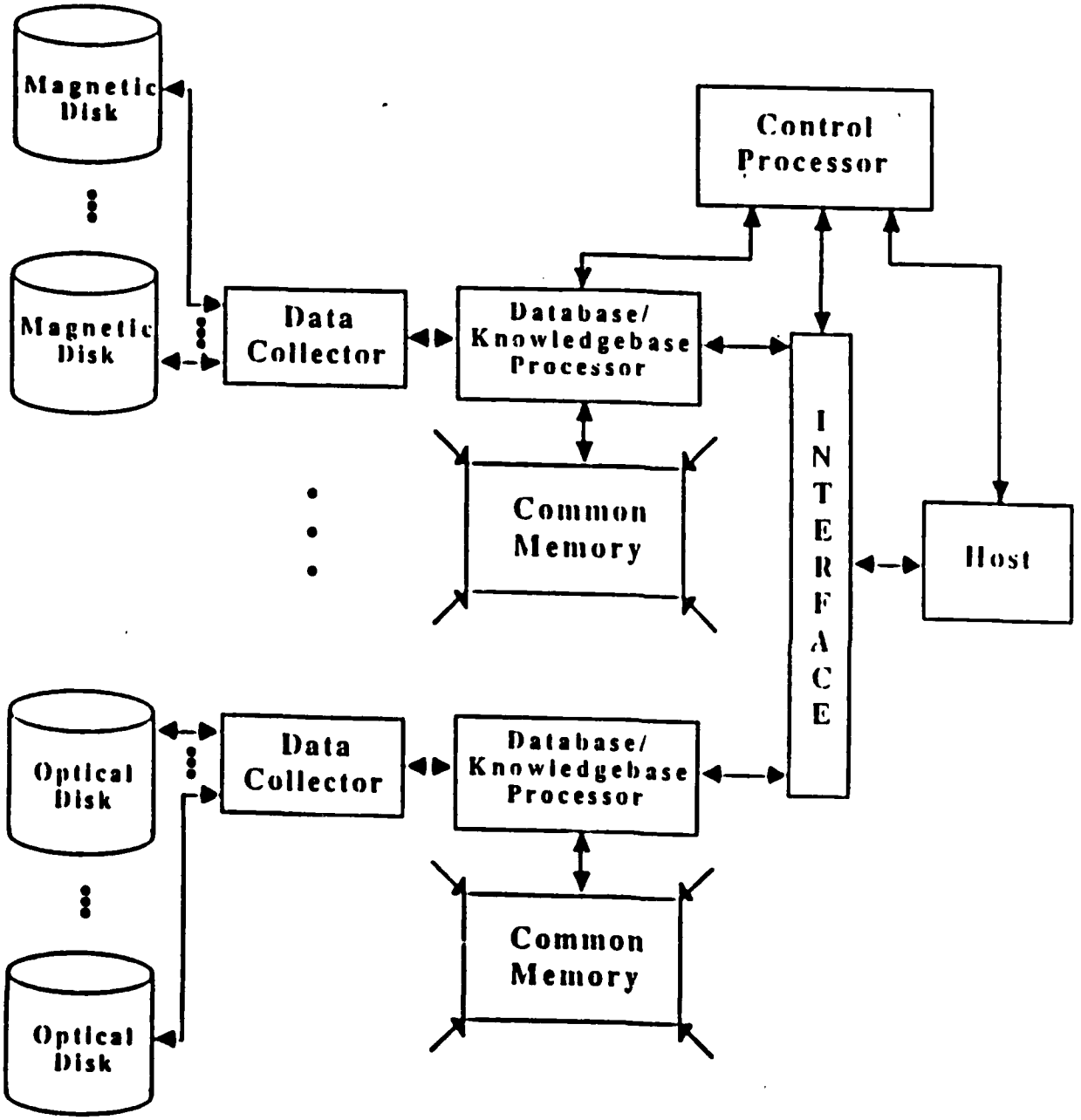


Figure A.4 Very Large Knowledgebase System

A.10 REFERENCES

Ahuja, S.R., C.S. Roberts, "An Associative/Parallel Processor for Partial Match Retrieval Using Superimposed Codes," 7th Annual Symp. Computer Architecture, pp. 218-227, May 1980.

Berra, P.B., S.M. Chung and N.I. Hachem, "Computer Architecture for a Surrogate File to a Very Large Data/Knowledge Base," to appear in Computer Magazine, March 1987.

Berra, P.B., S. Chung, and N. Hachem, "An Analysis of Surrogate File Structures for Very Large Knowledge Bases," draft technical report submitted to RADC.

Berra, P.B., E. Oliver, "The Role of Associative Array Processors in Data Base Machine Architecture," IEEE Computer, March 1979.

Berra, P.B., N. Troullinos, "Optical Techniques in Knowledge and Data Bases-Overview and Future Research Directions," submitted to Computer Magazine, July 1986. status: reviewed and now under revision.

Colomb, R.M., Jayasooriah, "A Clause Indexing System for PROLOG Based on Superimposed Coding," The Australian Computer Journal, Vol. 18, No. 1, February 1986.

Fagin, R., J. Nievergelt, N. Pippenger and H.R. Strong, "Extendible Hashing-A Fast Access Method for Dynamic Files," ACM Transactions on Database Systems, Vol. 4, No. 3, pp. 315-344, September 1979.

Garg, A.K., C.C. Gotlieb, "Order-Preserving Key Transformations," ACM Transactions on Database Systems, Vol. 11, No. 2, pp. 214-234, June 1986.

Hughes, K., "Interfacing Prolog with a Relational Database System," draft paper 1986.

Lee, D.L., "A Word-Parallel, Bit-Serial Signature Processor for Superimposed Coding," International Conference on Data Engineering, IEEE-CS, February 5-7, 1986.

Appendix A-1

Computer Architecture for a Surrogate File to a Very Large Knowledge Base

Computer Architecture for a Surrogate File to a Very Large Data/Knowledge Base

P. Bruce Berra, Soon Myoung Chung, and Nabil I. Hachem

Syracuse University

Special architectures that process surrogate files can greatly reduce the access time to very large data and knowledge bases. Also, some relational operations can be performed on the same file.

This article presents techniques for managing a very large data/knowledge base to support multiple inference-mechanisms for logic programming. Because evaluation of goals can require accessing data from the extensional database, or EDB, in very general ways, one must often resort to indexing on all fields of the extensional database facts. This presents a formidable management problem in that the index data may be larger than the EDB itself. This problem becomes even more serious in the case of very large data/knowledge bases (hundreds of gigabytes), since considerably more hardware will be required to process and store the index data. In order to reduce the amount of index data considerably without losing generality, we form a surrogate file, which is a hashing transformation of the facts. Superimposed code words (SCW), concatenated code words (CCW), and transformed inverted lists (TIL) are possible structures for the surrogate file. Since these transformations are quite regular and compact, we consider possible computer architectures for the processing of the surrogate file. We discuss the use of associative memory methods, as well as a back-end system that we are developing, in order to illustrate how nonsequential computer architectures can be used to advantage in solving this problem. Finally, we consider how one might perform relational operations on the surrogate file rather than on the full data.

Background perspectives

One of the most rapidly growing fields of artificial intelligence is expert systems. Currently, expert systems exist or are being constructed for medicine, business, and national defense. These systems are composed of a knowledge base of rules and facts and an inference mechanism that uses the knowledge base to respond to queries posed by users. The objective of such systems is to capture the knowledge of experts in particular domains (such as aircraft engine maintenance, blood diseases, computer configuration control, and financial analysis) and make it generally available to nonexpert users. At present such systems focus on narrow domains and have small knowledge bases; they are thus limited in their applications.

As these systems are expanded and more widely used, their knowledge bases will become more difficult to manage. The intensional database of rules will become

March 1987

0018-9162/87/0300-0025$01.00 © 1987 IEEE

large enough to present a formidable management task in itself. But the major management activity will be in the access, update, and control of the extensional database of facts.

The facts may occupy hundreds of gigabytes and in future serve several inference mechanisms, such as logic, frame-based, production rules, or semantic networks.

Much of our current research concerns the development of special computer architectures to access, update, and control very large knowledge bases that support multiple inference mechanisms for logic programming. The details of logic programming are widely known and will not be discussed here. However, we need to provide a simple example in order to set the stage for the problem of very large knowledge bases that we are addressing in our research.

Consider the following simple logic programming problem:

1. grandfather(x,y)—father(x,z), parent(z,y)
2. parent(x,y)—father(x,y)
3. parent(x,y)—mother(x,y)
4. father(pat, tiffany)—
   father(don, louise)—
   •
   •
   •
5. mother(mary, louise)—
   mother(lisa, tiffany)—
   •
   •
   •
6. —grandfather(x, tiffany)

The first three clauses form the intensional database of rules for this problem. The next two sets form the extensional database of facts regarding fathers and mothers. The last statement is the goal (or query). To solve the problem (satisfy the goal), we must find the names of the grandfathers of Tiffany. For this we search the father and mother facts on the second argument-position, finding values for the first argument-position that can be used later. Thus, we need to find Tiffany's mother and father before finding her grandfathers. If we ask a similar but slightly different query,

—grandfather(ray, y)

we search only the first argument of the father and mother facts in attempting to satisfy it.

Consider the fact type (relation) shown below:

r(a, b, c, ..., n).

In the general case one would want to access a relation on the basis of any subset of the arguments of the relation depending upon how the query was stated. This problem then becomes a special case of the partial-match retrieval problem: We are given a specific relation name and values for a subset of its arguments; we desire all tuples that satisfy the search criteria. These tuples are then returned to the inference mechanism for further processing.

For very large knowledge bases, how can we obtain the desired facts in the shortest time possible? If we assume that any subset of the attributes of a relation is equally likely to form the basis for a query, then we will probably have to index on all attributes in order to ensure reasonable performance. However, if we have 500 gigabytes of fact data, we could have 500 or more gigabytes of index data, an enormous amount.

We are currently trying to solve this problem in two ways. The first is to develop special hardware to process surrogate files*; these files can allow efficient access to the extensional database. The second is to consider optical techniques that can potentially increase input/output rates by orders of magnitude and thus speed access to the extensional database. This article concerns the first approach.

With the surrogate file approach, we transform the original data and then use it as an index to gain access to the EDB. This collection of transformed data is called the *surrogate file*. Well-chosen hashing functions can yield a surrogate file about 20 percent as large as the original data. We have analyzed concatenated code words, superimposed code words, transformed inverted lists, and combinations of these as basic structures for the surrogate file. We have simulated these structures and have obtained results that substantiate the 20 percent claim given above. We have considered associative memory and back-end system approaches or architectures for the surrogate files. Finally, we note that some relational operations can be performed on the surrogate file rather than on the extensional database.

Surrogate files

We assume that many different relations with varying degrees and cardinalities exist in the very large extensional database that we are considering. Furthermore, we assume that the tuples are stored in such a way that one first accesses the relation and then a particular tuple using a unique identifier. We obtain the name of the relation and values for a subset of its arguments from the logic programming process when it requests data. We obtain the unique identifier from processing the surrogate file. Thus, the storage structure for the facts themselves would be very simple and a method such as extendible hashing[2] would guarantee retrieval of a given fact in at most two disk accesses.

The principal techniques that we have considered for the construction of the surrogate file include concatenated code words (CCW), superimposed code words (SCW), combinations of CCW and SCW, and transformed inverted lists (TIL). We will use CCW and SCW to illustrate the ideas.

Suppose we have a relation called BORDERS:

BORDERS (Country 1, Country 2, Body of Water)

For a particular instance,

BORDERS (France, Italy, Mediterranean)

we would first hash the individual values,

H(France)	H(Italy)	H(Mediterranean)
↓	↓	↓
100...01	010...00	110...00

concatenate them, and then attach a unique identifier to obtain the CCW

100...01 | 010...00 | 110...00 | 00...01

where the vertical lines show the boundaries. The SCW[3] would be formed as follows:

H(France)	—100...01
H(Italy)	—010...00
H(Mediterranean)	—110...00
	110...01 | 00...01

with the binary representations logically ORed together. The unique identifier is attached as shown.

Thus, a surrogate file would exist with an entry for each fact in the EDB and the individual entries would be linked to the facts by their unique identifiers.

As illustrated in Figure 1 (using SCW), if we want to retrieve a fact based upon one or more arguments, we would pass each of the arguments through the same hashing function and OR their fixed-length binary representations together in order to generate the query code word

* The term "surrogate file" dates back to early work in information retrieval. Other terms, such as *signature files* and *descriptor files* have also been used to describe similar structures. See Faloutsos[1] for a review of various methods for accessing textual data.

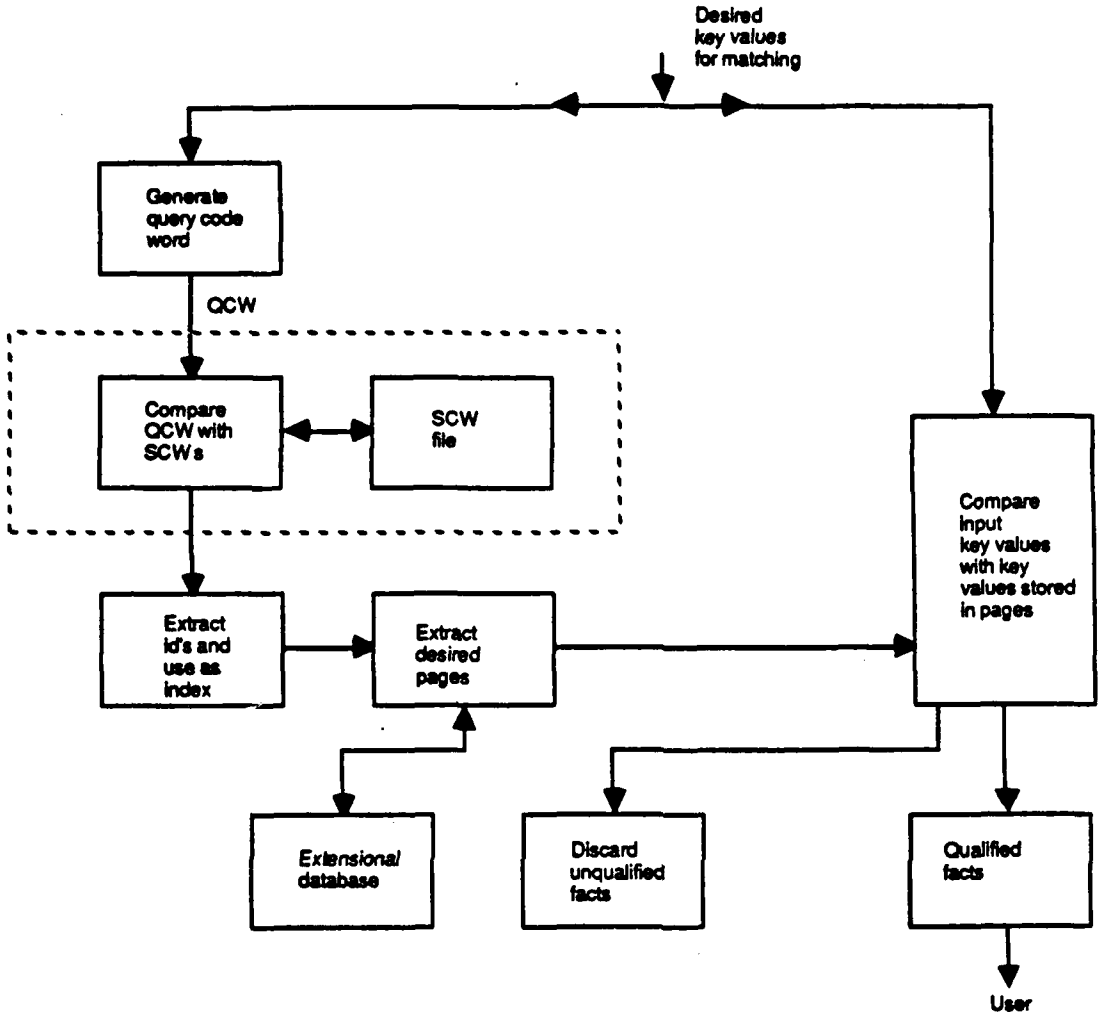


Figure 1. Partial-match retrieval using superimposed code words (SCW).

(QCW). We would then compare the QCW with all of the SCW for that relation, seeking matches on the ones of the QCW. All of the SCWs that had ones in the same positions as the QCW would be possible qualifiers. The id's of the possible qualifiers would then be used to retrieve the necessary facts. The input key values would then be compared with the respective key values of the retrieved facts in order to determine possible matches. If the desired fact (or facts) exists within the EDB, it will be found. However, a number of facts may be retrieved that are not the desired facts; these are called *false drops*. (The use of SCW for logic programming has also been discussed by Wise and Powers.[4])

Concatenating code words gives one extreme in word length while using SCW gives the other. The concatenation method almost ensures (subject to the quality of the hashing function in avoiding collisions) that the unique identifier(s) obtained is indeed the one(s) desired. However, it does so at the expense of a longer word length. The method of SCW reduces the word length, but at the expense of increasing the number of facts retrieved that are not the desired ones (false drops).

It may be that there is a point between these two methods that is optimal. For instance, Lloyd et al.[5,6] selected bit values from various positions of the binary representation of the arguments in each fact and concatenated them to form a code word. Also, we have explored a method that is a hybrid of the SCW and CCW methods. The resulting code word has unique portions for each argument and nonunique portions that represent the ORing of two or more arguments. While this method reduces the length of the code word, it does however increase the probability for false drops over that of CCW.

We have also investigated the use of transformed inverted lists. We form a TIL file in the following way: All of the values

for the first attribute position in a relation are passed through a hashing function to obtain the transformed values. Unique identifiers from each of the facts are associated with the respective transformed values. This will form the part of the TIL file for the first attribute. This process is continued for each of the other attributes in the relation, thus forming the entire TIL file. To access the TIL file with transformed values for attributes i and j, we need only search the portions of the TIL file with those values and then intersect their respective lists of unique identifiers. Thus, for few search values this technique is very good, but for many search values it is relatively poor, since more lists need to be intersected.

Surrogate file size

In this section we present the results of several simulations run for the purpose of comparing the storage requirements for SCW, CCW, and TIL files. The surrogate file size is determined as a percentage of the size of the EDB. Since tuples for relations are stored separately, the values given in this analysis apply equally well to the entire surrogate file or to a relation within a surrogate file. We assume that we are dealing with relations.

In the simulations we used two different values of relation size, 10^5 and 10^7 bytes. In each of the plots (Figures 2 through 5) the size of the surrogate file is given as the dependent variable and expressed as a percentage of the EDB. In Figure 2 we plot the size of the surrogate file as a function of the number of attributes (A_r) in a relation for SCW. In this figure, the average number of arguments in a query (R_q) takes on two values (one and two) and the number of false drops (FD) is equal to one. The value for R_q is set by the surrogate file designer depending upon the expected number of arguments contained in a query for the application under consideration. This establishes the size of the surrogate file. Queries with a lesser number of argument values than the design value would in general take more time to process and those with a greater number would take less time to process relative to the design value. The change in performance results from a greater or lesser number of false drops.

In Figure 3 we use the logarithm of the average redundancy ($\log_2 C_i$) as the independent variable for the CCW case. We have assumed that the argument values for the various attributes in the relation are not unique. In fact, in some cases they may have only a few values, thus yielding a very small surrogate file. For instance, there are very few values for the attribute color. With CCW, the size of the surrogate file is independent of R_q.

In Figure 4, we examine the effect of using a TIL approach to the structuring of the surrogate file. The particular approach that we used is a three-level index similar to that described by Cardenas,[7] except for first transforming the argument values. In Figure 5 we compare all three techniques. Note that for larger values of redundancy, all of the techniques yield a surrogate file size of less than 20 percent of the EDB

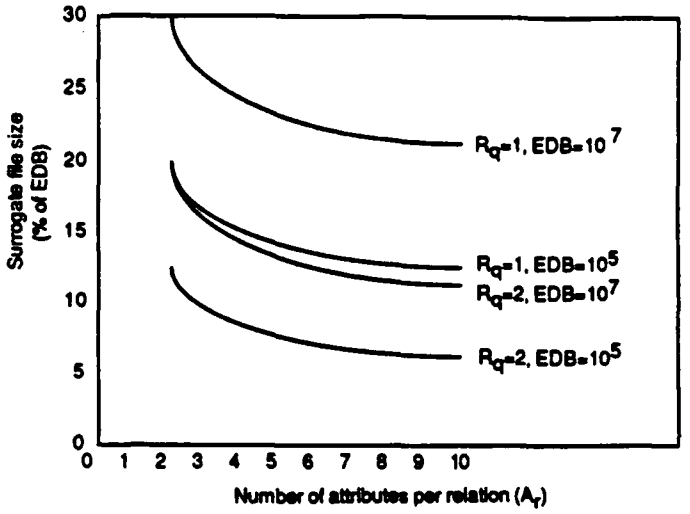


Figure 2. Surrogate file size for SCW.

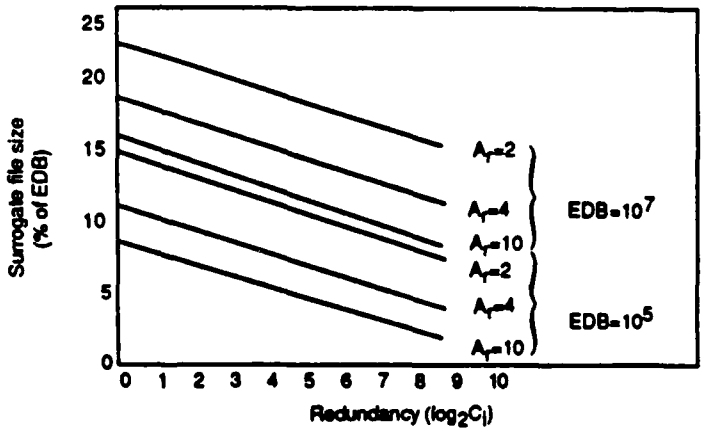


Figure 3. Surrogate file size for CCW.

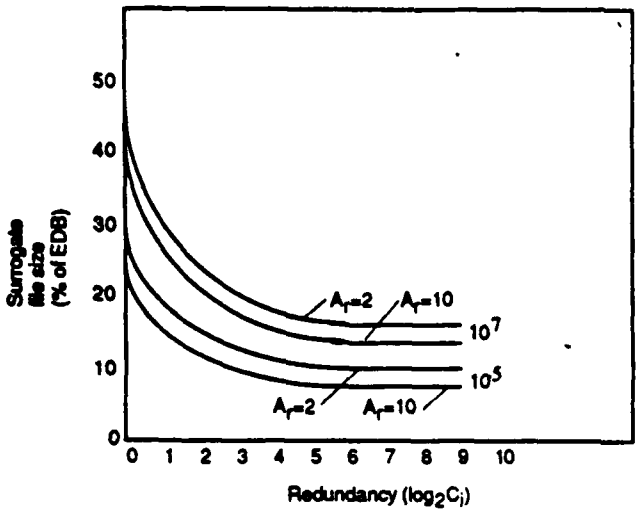


Figure 4. Surrogate file size for TIL.

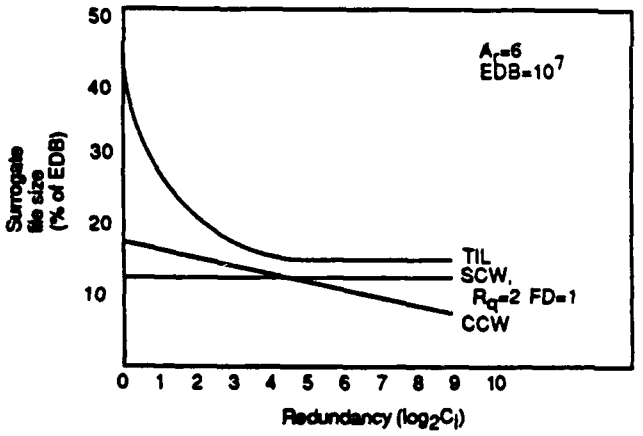


Figure 5. Surrogate file size comparison.

when the EDB is 10^7 bytes.

From Figure 5 one might conclude that CCW and SCW dominate TIL. However, with different parameter values the relative position of the curves is such that TIL is not dominated by CCW and SCW. Also, one must consider other factors, such as retrieval and update of the surrogate file, retrieval of the facts from the EDB, and the architecture for processing surrogate files, prior to selecting its structure. All of these factors taken together, as well as a knowledge of the application, are needed for the design of the system. That is what we are now considering in our research.

Updating

With regard to updating, a varying amount of work is required depending upon whether SCW, CCW, or TIL structures are used. To add a new tuple to a relation, we must generate the individual SCW or CCW, attach the unique identifier to both the tuple and the code word, store the tuple in the EDB, and append the code word to the code word subfile for that relation. For TIL-structured files, the list for each attribute of the relation must be processed, either by adding a unique identifier to the list of current unique identifiers for that value or by adding a new value and unique identifier to the list. The new tuple must then be added to the EDB.

To delete a tuple with the SCW and CCW methods, we must find and delete the entry in the surrogate file as well as in the relation. If TIL are being used, an entry in each list as well as the EDB must be found and then deleted.

When one changes the value of a field, SCW requires that a new code word be generated and the old one deleted. For CCW the change need only be made to the portion of the code word in question. With TIL files, only the entry in the list being changed need be processed. Of course, the value in the EDB must be changed.

Associative memory

We now consider processing the surrogate file by an associative memory when the file is constructed using SCW, CCW, or TIL. Recall that under our previous assumptions all partial-match retrievals will be performed using the surrogate file. Thus, retrieval of the facts will take only two disk accesses given the unique identifier.

Associative memories can perform many operations on array resident data, including exact match, maximum, minimum, and Boolean operations. These devices are fast because they use content addressing and parallelism, but they are generally costly, rigid in data format, and input/output bound.[8] Nevertheless, when associative memory processes the surrogate file, one of these disadvantages no longer matters and another is mitigated. That is, the format of the surrogate file is highly regular and maps very well into the associative memory. Also, the surrogate file is compact compared to the data being considered, therefore less I/O is required.

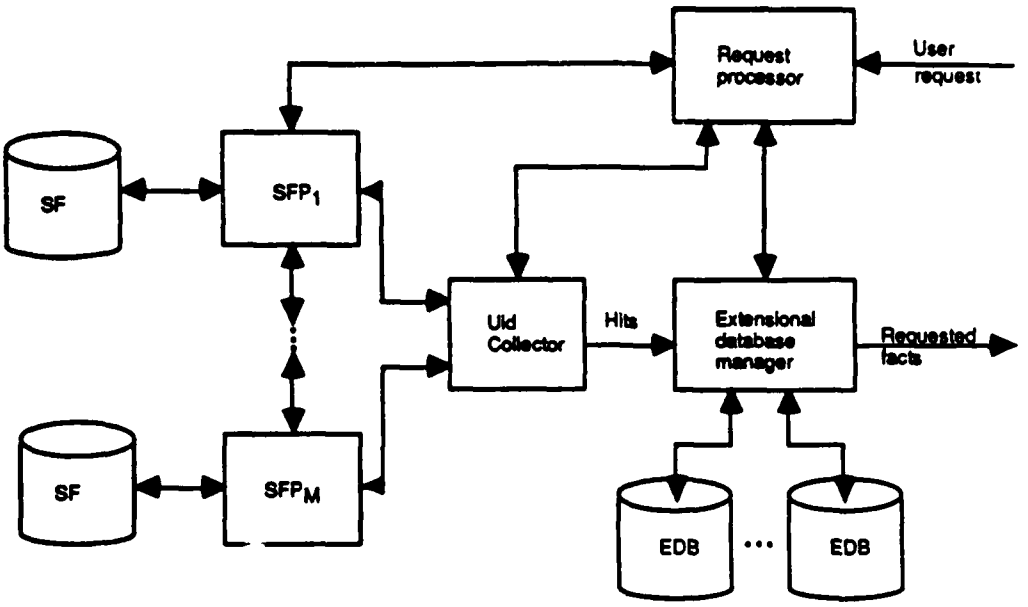


Figure 6. Back-end system for fact management.

In the case of SCW, research by Ahuja and Roberts[9] indicates that the mapping between an SCW file and associative memory is very good. The array resident data is processed by an exact-match search using the QCW as the search argument. The unique identifiers for all of the SCW that matched the QCW on the basis of ones would be sent to the fact manager for retrieval and further processing of the facts.

The processing of a CCW file is similar. The QCW has "don't cares" (dc) in positions with unknown values and a known bit pattern in the positions of interest. The results of an exact match search on the fields of interest are ANDed together and the selected unique identifiers sent on.

To process TIL files using an associative memory, one forms subfiles of unique identifiers for each value in each argument position, since these values will generally not be unique. Each distinct value will have an attached pointer that points to the list of unique identifiers for the facts containing that value. The processing of the TIL file will require less transfer of data to the associative memory for searches with few arguments, but will require intersecting the lists of unique identifiers before passing the results on. However, if many arguments are given in the request, considerable processing will be required.

Back-end system

Shown in Figure 6 is a back-end system for the management of a very large extensional database of facts. The system will also manage many intensional databases not shown on the diagram. We assume that there are many gigabytes of fact data on the EDB disks. Likewise, there are several gigabytes of surrogate file data stored on the SF, or surrogate file, disks. Since we have assumed the relational model, we will store the facts by relation and then by tuple-unique identifier within relations.

The surrogate file will actually consist of many subfiles, one for each relation. Each subfile will be distributed uniformly over all of the SF disks in order to ensure maximum parallelism in disk accessing. As an example, suppose that the surrogate file is constructed with concatenated code words and the user is interested in retrieving fact data given some subset of values from a particular relation. The query code word would be constructed in the request processor, using the proper hashing function and considering the positions of the values within the relation. The QCW would then be broadcast to all of the surrogate file processors (SFP) to be used as a search argument.

As in the previous section, one can think of the SFP as an associative memory with the QCW as the search argument. However, the full capabilities of the associative memory are not needed for these operations. There is no point in providing more capability in the SFP than is required to process the surrogate file data at disk rates. Thus, the SFP can be rather simple, since all it needs to do is compare the QCW with each CCW at disk rates and strip off the unique identifiers. As soon as any unique identifiers are found by the SFP, they can be sent to the collector and quickly passed on to the extensional database manager (EDBM) for processing, thus providing some level of pipelining. The EDBM retrieves the facts, compares them with the search criteria to insure that a collision has not occurred, puts them in blocks, and sends the blocks to the logic programming inference mechanism.

One follows a similar process if the surrogate file is constructed with SCW. The main difference is that extra processing would be required by the EDBM because of false drops. Other work regarding processors for SCW's can be found in Lee[10] and Colomb and Jayasooriah.[11]

If the surrogate file is composed of TIL, the processing will be slightly different and an additional hardware unit will have to be added to the system shown in Figure 6.

Each list would be distributed uniformly over the SF disks. Assume, for example, that a user's request requires access to only two lists. The first list is input to the SFPs and checked for an exact match. (Note that the SFP is just a simple comparator in this case.) The unique identifiers are stripped off and sent to the collector. The second list is processed in a similar way. However, now the two lists will have to be intersected. Thus, we will need an additional block between the collector and the EDBM, called the intersector.

In this discussion we have tried to show how such a system might retrieve facts from a very large extensional database. Obviously, many more issues need to be addressed, such as updating, integrity, and collisions.

Relational operations on the surrogate file

A serendipitous benefit obtained from using surrogate files is that some relational operations can be performed on the surrogate file rather than on the relations themselves. This could make relational operations much faster and increase the system's performance.

Consider a surrogate file constructed with CCW. To select on a particular field, one uses the surrogate file processor to do a partial match on just that field. Extending this slightly, clearly the SFP is simply a selection mechanism. With a little extra hardware, the SFP could be called the *selector*.

Consider a join using the hash join algorithm. Since the surrogate file already consists of hashed values, we need only assign the portion of the code word that represents the join variable and the associated unique identifier to buckets. Then, based on matching within each bucket (which can be done in parallel), pairs of unique identifiers can be sent to the EDBM for final verification. This would work for the equijoin join, but an order-preserving hashing function[12] would be needed for other joins.

It appears that there would be no advantage to performing projections on the surrogate file. Thus, the EDBM would have to store the tuples in such a way that projections could be done directly on the tuples of the relation under consideration. This could be done by treating each relation independently, with its own extendible hashing system. Storing relations this way would also aid other operations that

access many or all tuples in the relation.

Finally, range queries could be performed on the CCW file given a suitable order-preserving hashing function, albeit with some additional processing, such as sorting.

In the case of SCW, it would be difficult to perform most relational operations on the surrogate file. In the case of TIL files, selections and hash joins could be performed, but projections would have to be performed on the tuples themselves. Range queries could also be performed given a suitable order-preserving hashing function and some additional processing.

We have tried to sketch some ideas on the development of computer architectures for processing of surrogate files of a very large data/knowledge base. In addition, we have pointed out how surrogate files might be used to perform relational operations. We believe that many logic programming and relational model ideas will be integrated over the next few years, thus it may well be possible to perform significant logic operations on the surrogate files as well. Our challenge now is to design the hardware that will make these ideas a reality.□

Acknowledgments

This work was supported by the Air Force Systems Command, Rome Air Development Center at Griffiss Air Force Base in New York, and the Air Force Office of Scientific Research at Bolling AFB in Washington DC under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

References

1. C. Faloutsos, "Access Methods to Text," *Computing Surveys*, Vol. 17, No. 1, Mar. 1985, pp. 49-74.
2. R. Fagin et al., "Extendible Hashing—A Fast Access Method for Dynamic Files," *ACM Trans. Database Systems*, Vol. 4, No. 3, Sept. 1979, pp. 315-344.
3. C.S. Roberts, "Partial Match Retrieval via the Method of Superimposed Codes," *Proc. IEEE*, Vol. 67, No. 12, CS Press, Los Alamitos, Calif., Dec. 1979, pp. 1624-1642.
4. M.J. Wise and D. Powers, "Indexing Prolog Clauses via Superimposed Code Words and Field Encoded Words," *1984 Int'l Symp. Logic Programming*, CS Press, Los Alamitos, Calif., Feb. 1984, pp. 203-210.
5. J.W. Lloyd, "Optimal Partial-Match Retrieval for Dynamic Files," *BIT 20*, 1980, pp. 406-413.
6. J.W. Lloyd and K. Ramamohanarao, "Partial-Match Retrieval for Dynamic Files," *BIT 22*, 1982, pp. 150-168.
7. A.F. Cardenas, "Analysis and Performance of Inverted Data Base Structure," *Comm. ACM*, Vol. 18, No. 5, May 1975, pp. 253-263.
8. P.B. Berra and E. Oliver, "The Role of Associative Array Processors in Data Base Machine Architecture," *Computer*, Vol. 12, No. 3, Mar. 1979, pp. 53-61.
9. S.R. Ahuja and C.S. Roberts, "An Associative/Parallel Processor for Partial Match Retrieval Using Superimposed Codes," *7th An. Symp. Computer Architecture*, joint publication of IEEE-CS and ACM, Los Alamitos, Calif., May 1980, pp. 218-227.
10. D.L. Lee, "A Word-Parallel, Bit-Serial Signature Processor for Superimposed Coding," *Int'l Conf. Data Engineering*, IEEE-CS, Los Alamitos, Calif., Feb. 5-7, 1986, pp. 352-359.
11. R.M. Colomb and Jayasooriah, "A Clause Indexing System for Prolog Based on Superimposed Coding," *The Australian Computer J.*, Vol. 18, No. 1, Feb. 1986, pp. 18-25.
12. A.K. Garg and C.C. Gotlieb, "Order-Preserving Key Transformations," *ACM Trans. Database Systems*, Vol. 11, No. 2, June 1986, pp. 214-234.



Soon Myoung Chung has been studying in the PhD program of the Electrical and Computer Engineering Department at Syracuse University, New York since September 1984. His current research interests include database machine design and digital image processing.

Chung received his BS degree from the Seoul National University, Seoul, Korea, in 1979 and his MS degree from the Korea Advanced Institute of Science and Technology, Seoul, Korea, in 1981. From March 1981 to June 1984, he worked for the Daewoo Engineering Co., Seoul, Korea in the areas of instrumentation and computers.



P. Bruce Berra is currently a professor of electrical and computer engineering and a member of the faculty of Computer and Information Science at Syracuse University. He previously held the position of professor and chairman of Industrial Engineering and Operations Research, also at Syracuse University.

Berra received BS and MS degrees from the University of Michigan and a PhD degree from Purdue University. He has served as the Editor-in-Chief of the IEEE Computer Society Press, on the IEEE Press Editorial Board, as a member of the Distinguished Visitor Program, as program chair and general chair of prior Data Engineering Conferences, as a member of the IEEE-CS Governing Board, and for several years on the Publications Board. He is currently Editor of the *Transactions on Software Engineering*, Editor of the *Transactions on Computers*, and a Distinguished Visitor for the Computer Society of the IEEE.



Nabil Hachem is presently working towards his PhD in the Department of Electrical and Computer Engineering at Syracuse University, New York. His current research interests include database machine design.

Hachem received his BS degree in electrical engineering in 1979 from the American University of Beirut, Lebanon. Between 1979 and 1985 he was a member of the technical staff at Ditacomm, Lebanon, where he occupied the position of Operations Department Manager.

Readers may write to Berra at the Dept. of Electrical and Computer Engineering, Syracuse University, Syracuse, NY 13244.

Appendix A-2

Optical Techniques in Data and Knowledge Base Machines

Optical Techniques in Data & Knowledge Base Machines

P. Bruce Berra

Nikos B. Troullinos

Dept. of Electrical and Computer Engineering

Syracuse University, Syracuse, NY 13244-1240

{berra,nicktrou}%syr-sutcase.csnet@relay.cs.net

March 1987

This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F306002-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

CONTENTS

Abstract

Introduction

Optical storage

Digital optical processing

Architectural issues

Optical Data/Knowledge machines

Abstract

Optical techniques for data storage are gaining wide acceptance in computer applications because of their very large data densities. Optical interconnections are beginning to be utilized in intrasystem communications because of their high bandwidth and lack of interference. Optical switching elements are under investigation because they offer the possibility of processing information in a completely optical manner. This paper addresses the role of optics in the field of data/knowledge bases given their computational and technological requirements; more general repercussions are also discussed.

Introduction

The task of collecting, accessing and maintaining data, in all its forms, is the main concern of database management. Over the years it has been established as one of the most vital computer applications. With the advances of technology and the ever increasing dependency on computers, these systems have expanded both in number and complexity and now encompass such diverse application areas as distributed, multimedia and CAD/CAM databases; as well as knowledge bases for expert systems. Hundreds of commercial Data Base Management Systems (DBMSs) are available, targeted for machines ranging from mainframes to personal computers.

Database systems are not without problems however. They are costly to develop and consequently consume considerable computing resources. But it is only fair to recognize that when we speak of problems and limitations we are often referring to the ever growing aspirations for more powerful systems at even lower costs. There are many respects in which current database systems need further development: the amount of data that can be stored is often insufficient, although well into the range of trillions of bytes for the largest applications; the response time can be slow, especially for complex transactions like context-sensitive searches or searches of unstructured data such as in full text retrieval systems and the interface to the user is not optimal despite powerful query languages, often of a non-procedural nature. The cost of acquiring and maintaining the hardware-software components of such systems is high, although the performance to cost ratio is being improved continuously.

Solutions to these problems demand much more than an alternative hardware approach. But *database machines*, i.e. computers with architectures and software optimized for database management, help in solving or easing some of the above limitations. Moving functions from software to hardware and performing as many as possible in parallel, are two directions which lead to performance improvements. Progress in electronic technology, especially with VLSI, has lowered the costs of logic and memory enough to make deviations from the classic general purpose architectures attractive. Unfortunately, the main obstacle, access time for data in magnetic secondary storage has remained essentially constant despite the dramatically increased storage capability of the devices themselves.

If one abstracts from the qualities of the proposed or implemented database machines the following are present or desirable: very large storage capacity; use of specialized structures for the disk I/O; memory hierarchy with large data cache; utilization of parallelism and content addressable (associative) memories; special purpose architectures for performing well defined primitive functions like selection, joining or sorting and, finally, operating systems of suitable functionality and performance.

Commercially, the field of database machines is not yet mature, with only a few products on the market and various research efforts going on at university and industrial settings [Neches86]. The issues involved in designing any type of new architectures are complex and all encompassing, and the traditional architectures are so deeply rooted (or well serving) that progress is bound to be rather slow.

The field of optical computing is currently receiving considerable research attention. This is due primarily to the speed and parallelism inherent in light waves

and the large storage density achieved in optical disks. With the requirements for database management as given above it is natural to look to optics for possible solutions. Optical disks, as discussed in the next section, have enormous capacities and although are currently characterized by low data transfer rates they have the potential for order-of-magnitude increases in transfer rates beyond magnetic disks.

The inherent speed and bandwidth of optics has already resulted in major advances in telecommunications because of optical fiber technology. The advantages of optics are beginning to be felt in multiple processor commmunication and will likely affect future designs even to the interchip or intrachip levels. The development of optical processors is also receiving considerable attention. All these developments taken together may well have a significant impact on data and knowledge bases as we shall discuss in subsequent sections of this article.

Optical storage

Storage is a sort of "raw material" for data/knowledge base systems. It is required in great quantity and at the lowest cost possible. Technology has done very well so far in keeping up with (and fueling) the demands; the figures stating the decline of cost per stored bit are always very impressive (Fig 1).

	MOS RAM	*Magnetic Disk*	*Optical Disk*
Capacity	1 Mbyte	1 Gbyte	10 Gbyte
Access time	100ns	20ms	100ms
Cost	$100	$10000	$10000
Volatile	Yes	No	No
Erasable	Yes	Yes	No*
Comparison			
Access time	1	2×10^5	10^6
Cost/bit	100	10	1

* not presently

Fig. 1 **Order of magnitude figures for storage /memory elements**

The widespread appearance of optical storage can be traced to the introduction of video laser disks nearly a decade ago. In its simplest form, a beam of light detects the presence (or absence) of pits on a revolving reflective layer and responsive servomechanisms are employed for tracking and focusing. The end result is data storage at areal densities an order of magnitude higher than those of the best magnetic hard disks and hundreds of times more than those of flexible disks. These properties are even more impressive if we consider the fact that imaging is done from a considerable distance (order of millimeters) and we can

dispense with extreme mechanical tolerances and the super clean environment of high performance magnetic disks [Hoagland85], [Laub86] (Fig 2). This implies cheap removability, an almost ideal solution to the problem of voluminous magnetic storage: data backup. Another implication is the development of automatic changers or "jukeboxes" with an aggregate capacity in the hundreds of gigabytes.

	Magnetic	*Optical*
Head/medium gap	0.1µm	1000µm
Substrate	Ultra-smooth Al	Al,glass,polymer
Medium thickness	< 1µm	< 1µm
Encapsulation	No	Yes
Environment	Sealed	Open

(Hoagland 85)

Fig. 2 Magnetic and optical storage devices

The development of optical disks will almost surely follow the trends present in the well established and mature magnetic disk industry: high performance, state of the art devices from a few manufacturers for the mainframe and advanced applications market, and low-cost, low-end but respectably performing products for a commodity-like market of high volume and intense competition.

Technical and cost limitations have dictated three types of optical storage that can be compared to the ROM, PROM and RAM memories of semiconductor technology. *Read-only* optical disks have their contents fixed at production time, usually by stamping from a master, and will be used mainly for distribution, in a machine-readable form, of materials previously printed or remotely accessed. The strongest contender for the immediate future is marketed under the name of CD-ROM and is a direct descendant of the audio compact disk. It offers a storage capacity of

about 600 megabytes on a disk 120mm in diameter that can be mastered for approximately $3000 and can be replicated in quantity for $5 each [Chen86]. At least a dozen companies have models either in inventory or production. Prices for the drives range from $500 to $2000 and are bound to decline with massive demand and availability. Having randomly accessible information at the volume and cost that is provided by CD-ROMs opens up a new world of applications. Both professional and consumer markets can be well served and CD-ROMs may well prove to be the second coming of the home computer.

Write-once disks avoid the mastering and stamping steps when mass replication is not needed. Information can be recorded in a non-reversible way which makes this type of storage ideal for archival purposes. But even for operations that do not have an archival nature the extremely large capacity of the disk renders the non-erasability relatively unimportant. Their widespread acceptance is of course subject to the development of operating system support and standardization. Currently available optical disks are of this and the previous type with the largest and more expensive ones tending to be of the write once (or WORM: write once - read many times) type. There is little agreement on the mechanical or electrical characteristics and one of the few common features is the ability to interface with the IBM PC. In the near future it is likely that 10 gigabyte drives will be available for about $10000, a substantial saving over magnetic disk drives.

Finally, *read/write* erasable disks based on magneto-optic, phase change or mechanical deformation phenomena in a variety of materials may prove to be a superior counterpart to current, state-of-the-art, magnetic disks [Bloomberg85]. The 1990s may lead to disks with transfer rates improved by orders of magnitude

and acceptable access times in comparison to the fastest magnetic disks now available.

Optical storage will most likely complement but not replace magnetic storage. On-the-fly erasability without undue fatigue is the major technical problem which must be solved before the impact is widely felt. Standardization can neither be hurried nor indefinitely postponed without causing either a blocking of innovation or yet another interfacing nightmare. Even by conservative estimates the cost of having information available within the access time of a disk will drop dramatically. The natural outcome is to see applications greatly increasing their demands for storage.

Because of the increased demands, these improvements in storage technology will make corresponding advances in computational performance even more imperative. And this will remain true whether or not the technology used is electronic.

Digital optical processing

The development of the digital computer is so closely associated with the development of electronic technology we tend to think that they are inseparable. But the elementary operations are logical and could be performed by a variety of different technologies.

Research in the general area of information processing using optical techniques has been going on for nearly three decades. Recent advantages in optical bistabilities offer promises of a broader view of optical information processing. Traditionally, optical processing has been applied mainly to images and in an

"analog" way. But the possibility of optical logic and optical memory brings general purpose, digital optical computing closer to reality [Huang84],[Abraham83].

A digital optical computer, whether general purpose or more specialized (like data/knowledge base machines) has many potential advantages. In order to realize and weigh the different qualities of optics versus those of electronics we have to consider the factors limiting present day electronic computers. Clock period and pulse widths have shrunk in order to increase throughput. But the *bandwidth* needed for transmitting these signals intelligibly is very large. As a result interconnections in supercomputers have to use bulky, expensive, transmission lines increasing cost disproportionally.

A second problem is *clock skew*, i.e. the difference in arrival time of pulse fronts that originate from distant parts of a circuit. This skew of the input signals could potentially cause a gate to generate erroneous outputs. In order to avoid skew, interconnection length must be restricted and connections with different electrical lengths must be padded to compensate for the different intrinsic delays. Unfortunately, these problems are not alleviated by the use of VLSI because, as it is widely known, dimension scaling leaves the RC time unchanged, at least in a first order approximation.

Switching speed of the logic elements is another constraint. Silicon-based semiconductor logic has already approached the physical limits and the investigation of new materials like GaAs has resulted in high speed practical devices. Super-cooled devices (Josephson junctions) looked promising a few years ago but it seems that they pose many practical difficulties.

Optics may have some answers for these limitations. The large bandwidth, innate parallelism and non-interfering propagation offer mechanisms for overcoming communication problems. Switching times for recently demonstrated optical logic elements range from milliseconds to femtoseconds and although experts agree that it is too early to reach conclusions, there is hope for dramatically increased performance.

Given that we do not want to forgo the undisputable advantages of digital information processing any viable alternative to semiconductor devices must include:

(a) Elementary switching devices with (usually two, but multivalued logic may also be considered) stable states which can represent information encoded digitally, and

(b) Functional devices, built from the above, such as logic gates and memory cells integrated in dense and low cost packages.

It appears more than one optical phenomenon can be put to work. *Fabry-Perot cavities* and *quantum wells* are two of the structures that have been utilized so far with moderate success.

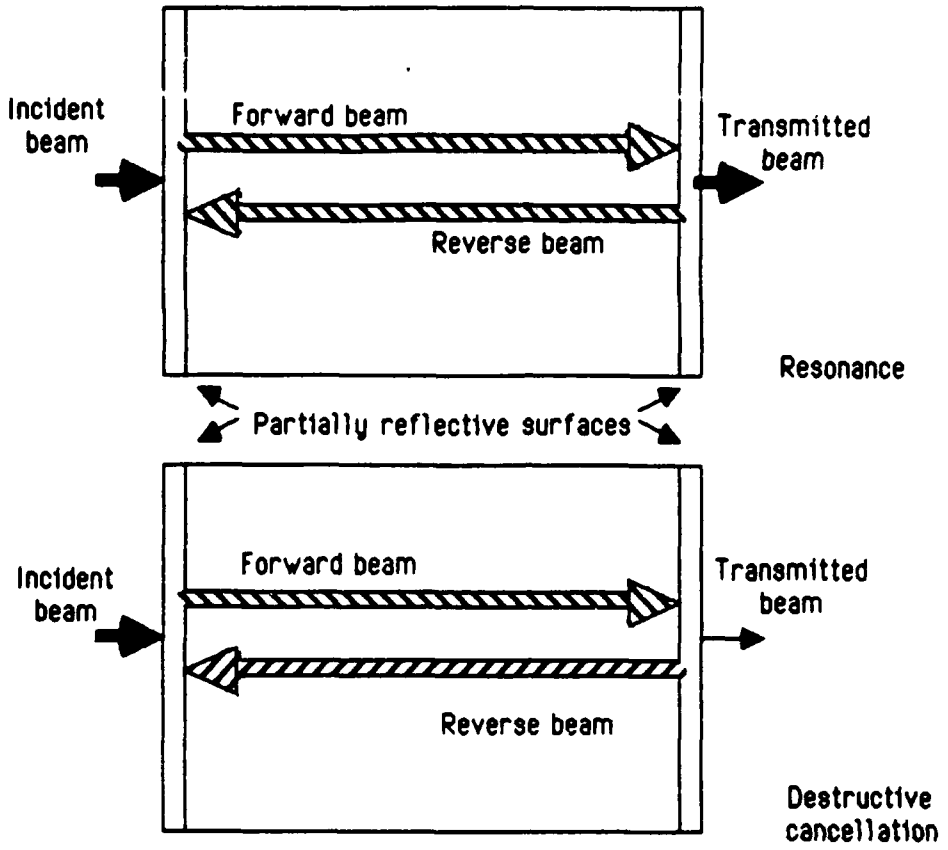


Fig. 3 Fabry-Perot cavity

A Fabry-Perot cavity or etalon (Fig 3) transmits incident light when the optical length of the cavity is an integral multiple of half-wavelengths. If the space between the two partially reflective surfaces consists of an optically nonlinear material (Fig 4) then the transmitted light intensity follows the familiar hysteresis curve (Fig 5).

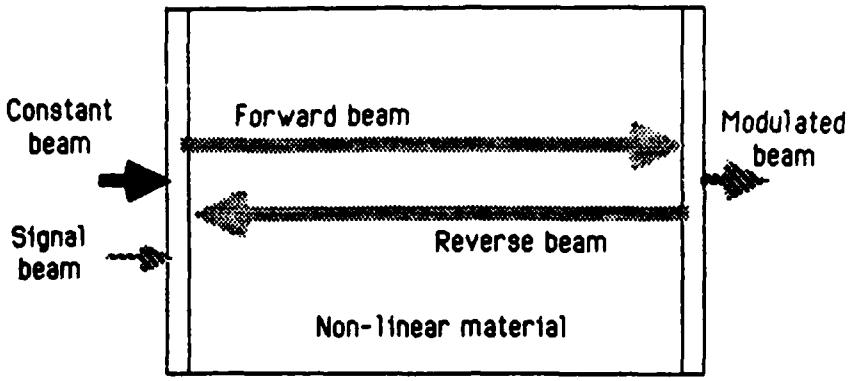


Fig. 4 **Optical amplification in a Fabry-Perot cavity**

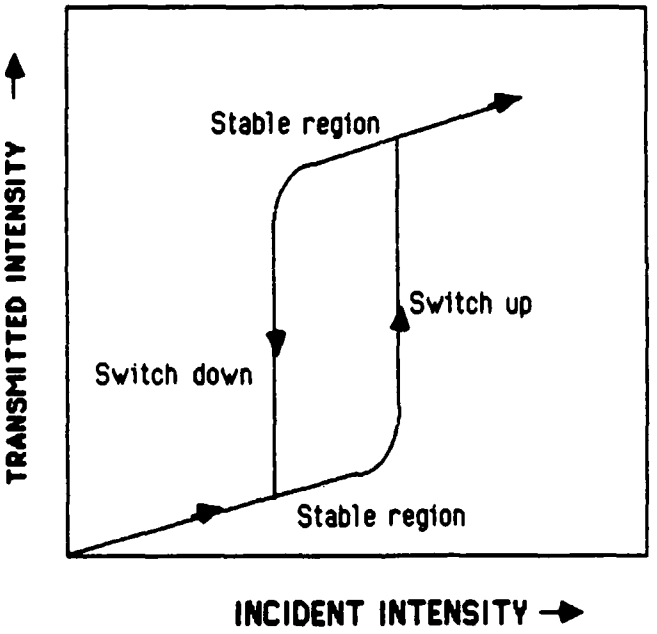


Fig. 5 **Effect of a non-linear material in.a Fabry-Perot cavity**

Quantum-well materials are formed by alternating extremely thin layers of two materials with different electron band gaps. The overall effect is nonlinearities which are far more intense than those that can be obtained using

intrinsic semiconductors. Two optical devices based on quantum well phenomena are known as SEED (self electrooptical effect device) and QWEST (quantum well envelope state transition) [West85].

The above cases must be further researched in order to determine whether they exhibit the required properties of alternative logic elements as given below:

- The input/output transfer function must have a shape similar to the ones shown in Fig 6. The one on the left is the necessary shape for transforming the sum of the intensities of two signals to the AND function and the one on the right is for the OR. For implementing inversion (negation) a descending symmetric curve is necessary.

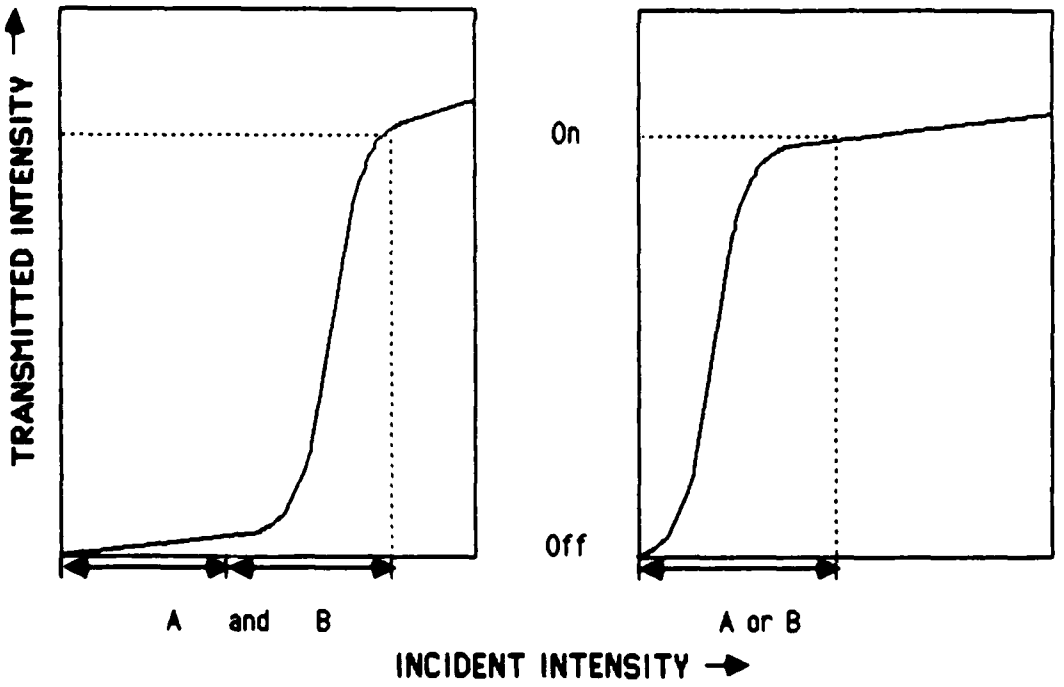


Fig. 6 Implementing Boolean Connectives

- The nonlinearity must have a gain of at least 4 - 10 times to accommodate fanout and distribution losses.
- Logic levels must be restored after each operation because, in contrast with traditional "analog" applications of optics, digital processing requires thousands of operations. This requirement is equivalent to saying that the logic elements must have three "ports", just like an electronic transistor.
- Phase variations (a unique problem due to the wave nature of light) should not affect the function of the logic element. For all possible device states reflections and cavity resonances should be controlled (the equivalent of electrical "impedance matching").
- Though in theory two-input gates are all that is needed, practical systems will be very difficult to design unless four or more mutually non-interacting inputs are provided.
- Tolerance to fabrication variations like doping densities, line widths and surface flatness are essential for low cost as is relative immunity to temperature.

The previous account does not exhaust the possibilities for optical logic. *Shadow casting* techniques (albeit based on optical to electrical conversions) have been proposed [Ichioka84]. Memory elements are at least in theory obtainable when amplifying non-linearities are combined in a setup that exhibits positive feedback.

Digital data, at least in the traditional character-based applications, is inherently one-dimensional. *Acoustooptic devices* are very good in accurately

transforming one-dimensional time-domain signals of extreme bandwidth (order of gigahertzs) to spatial perturbations, in effect "freezing" the input signal. Optical processing can then be applied in order to perform common operations like correlation and pattern recognition [Psaltis1984].

For two dimensional data, such as pictures, *spatial light modulators* are used to modify the intensity of a reference beam of light according to the intensity of an input beam [Mnatsakanyan79] [Fisher85]. Operations like amplification, negation and thresholding can be done in parallel for all points on the device surface. Unfortunately the time response of most known devices is in the order of milliseconds.

It is too early for a verdict to be reached concerning the prospects of optical techniques, especially if we are to take into account that the yardstick against which they are to be measured is the well matured electronic techniques. Smith and Tomlison [Smith81] reviewed optical devices and compared them to electronic ones. Although they were reluctant to speak of a digital optical computer they reported superiority of optical devices in the subpicosecond range of switching rates.

The switching power required for this ultra fast operation is rather high, with thermal transfer problems as a result, but they argued that this is not going to be a very serious obstacle. The reasons are, first, that devices with reactive nonlinearities reflect and do not absorb most of the incident light energy and, second, that the low duty cycle required if these devices are to interface with "slow" electronics greatly reduces the restrictions imposed by thermal dissipation.

The question as to which technology switches the fastest is only part of the overall problem. The communications ability of the technology is at least as important. It becomes evident as architectural questions come into play that an inability to communicate can quickly compromise any advantage a technology might have in terms of speed. To appreciate the severity of this problem one need only consider the rapid degradation of performance in multiprocessor systems when there is appreciable interprocessor communication.

From a simplistic viewpoint, the relative merits of electronics and optics seem to correlate to the properties of electrons and photons [Huang84]. Electrons can influence each other at a distance, so it is easy for an electrical signal to affect another one to perform switching; however, this interaction has unwanted side effects in the form of capacitance and inductance which complicate communications. Photons are the opposite. The absence of interaction gives optics the superior communication qualities but also makes it hard to perform switching. Nevertheless, switching has been achieved at energies comparable to electronics and this may eventually put electronics in a disadvantage.

Architectural issues

The hardware of electronic computers has made tremendous advances in the last 40 years going from vacuum tubes to very large scale integrated circuits. On the contrary, the architecture of computers and the programming languages used to instruct them have remained fundamentally the same because they implement the computing model named after John von Neumann. As a result the opportunities offered by VLSI and parallelism are not exploited and computer power is reaching the limits imposed by the physical laws of electronics.

The central problem of the von Neumann model is referred to as the *von Neumann bottleneck* and involves the performance limitations that result from the sequential, address-mediated communication between the CPU and memory. It is brought on by the limited number of interconnections that can be supported in a practical manner by an electronics-based technology [Huang84,85] [Jenkins85].

For example, if memory consists of N bytes then it is completely impractical to support N interconnections between the memory and the logic unit with wires (Fig. 7). Using address decoding the interconnections are reduced to $\log_2 N$ which is very practical since an immense amount of memory, e.g. 4 Gbytes, can be addressed with only a few, in this case 32, lines and a prodigious one, e.g. 16 G^2bytes with only 64. This is the classic space-time tradeoff.

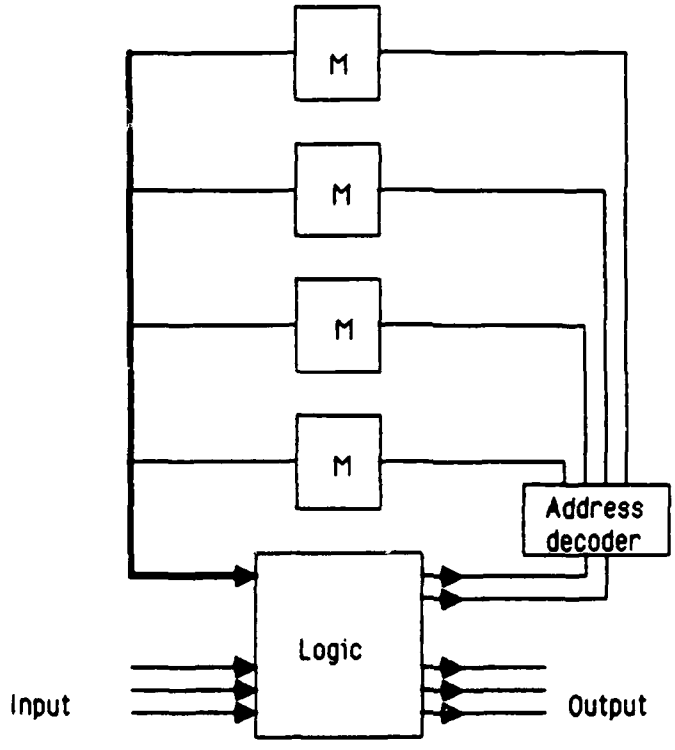


Fig. 7 Finite State Machine v. Von Neumann bottleneck

Constrictions similar to von Neumann's are also found at other levels. At the module level broadcast-bus structures are used for communicating between modules. At the chip level the limited number of external connections forces the use of time multiplexing. In contrast, the number of elements within the chip can be enormous and many applications would benefit from parallel signal communications.

One of the simplest models for computing, the classic finite state machine does not have this sort of problem. All storage elements are updated in parallel without the need for addresses (Fig. 8). In addition, the storage elements need not preserve their contents for more than one cycle. This may prove valuable in our quest for practical optical memories.

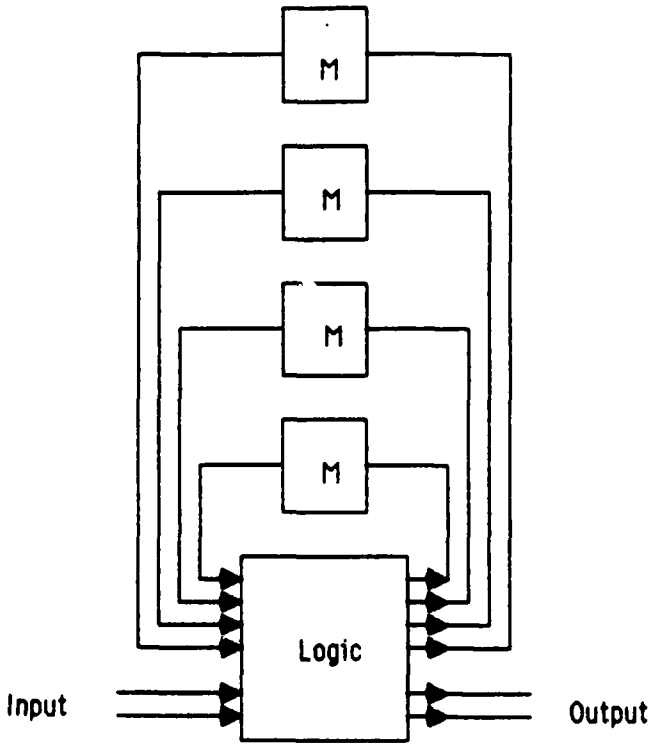


Fig. 8 Classic Finite State Machine

The difficulty with FSMs is that the number of connections between the combinational and memory units is very large. But optical techniques may already have an answer for this. Sawchuck and Strand [Sawchuck84] described an experimental system which was composed of two main components. A spatially parallel array of independent optical gates provided the logic and memory functions, and a computer generated hologram served as a beam-steering element to arbitrarily connect gate outputs to gate inputs in a free space setup (Fig 9). Their system had only 16 gates but it was built to demonstrate the concept by optically implementing a synchronous master-slave flip flop and a driving clock.

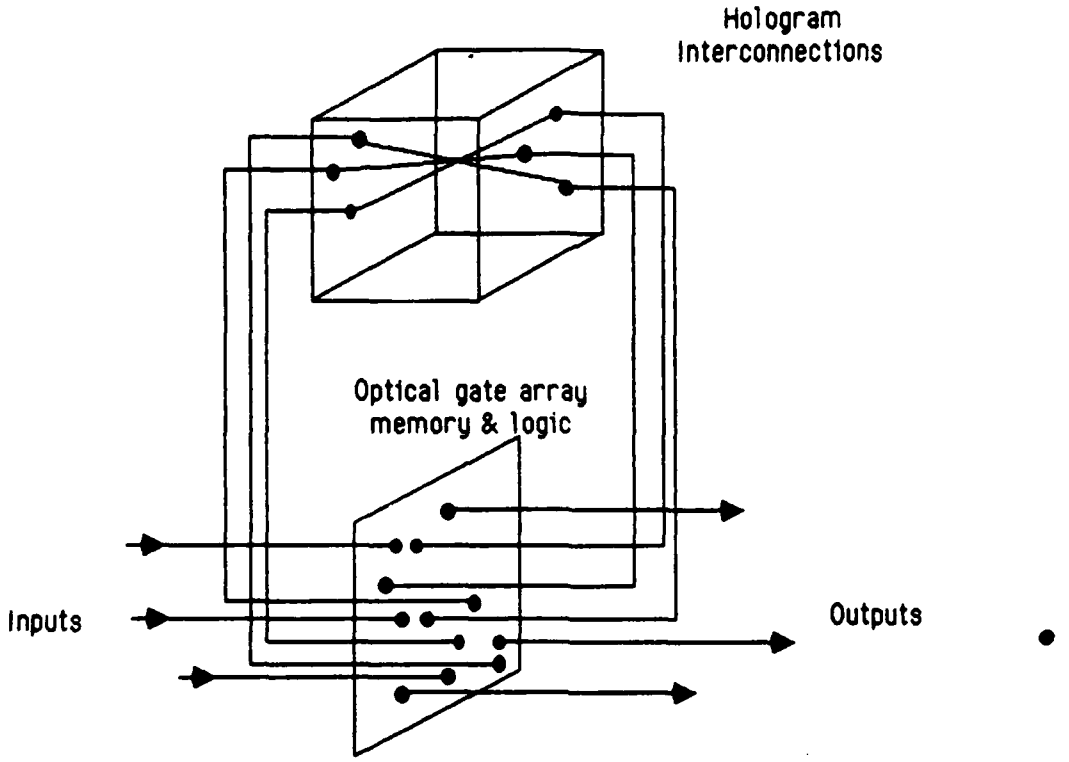


Fig. 9 Free-interconnection non von Neumann processor

It is not clear whether the above results can be extrapolated to systems with thousands or millions of gates . The previous authors state that arbitrarily *space-variant* interconnections are limited by current hologram recording techniques. On the other hand, for *space-invariant* interconnections, like the ones necessary for cellular logic, there is practically no limit even with today's technology.

Huang gave another example of an optical processor but he mentioned that it was meant more as an existence proof rather than as a practical design [Huang84]. He demonstrated the practicality of an exceedingly uniform and simple approach based on a functional logic block that can perform all sixteen Boolean connectives such as NOR, NAND, OR etc. These blocks, which are implemented by using only AND

gates, are grouped in pairs called *logic cells* so that both a signal and its complement can be simultaneously generated. Finally, pairs of logic cells can be grouped to form a *function/interconnection module* that can be programmed to either perform logic or implement various interconnection primitives. If the latching properties of optical gates (as propagation time approaches switching time) are used for storage and the modules are connected so that the output of the one is the input of the next, then an optical pipelined processor will be formed.

Even if totally optical processors are never realized, there are other areas in computer system architecture where optics will play a leading role. Two comprehensive papers describe the prospects of optical interconnections [Goodman84,85]. Currently, the advantages of photons as carriers of information are used to connect machines in local area networks with optical fibers. At a more detailed level, communication between modules of high-speed multiprocessor machines can benefit from optical propagation. Further down, chip to chip communication at the projected high rates could be done optically with less power and interference. Finally, within integrated circuits optics seem particularly well suited for distributing gigahertz clock signals.

Optics may also have solutions for dynamic interconnection needs. We are frequently shying away from implementing efficient parallel algorithms because of the global, random references they demand. *Crossbar switches* and *perfect shuffle* networks are two topologies that can be beneficially realized with optics [Bell 86]. Indeed, interconnections may prove to be optics' greatest contribution to computing.

One general observation that should be kept in mind concerning the architecture of optical computers is that an evolutionary approach might not be the

best. Optics has unique characteristics and qualities and these should be exploited by appropriate architectures, even if that implies that we have to widen our perception of computing machines.

Optical Data/Knowledge Base machines

More applied areas of research in the area of optical computing could take into account the idiosyncrasies of specialized architectures. Some problems may be simplified if a narrower view is taken. In knowledge and data base applications for instance, *selects*, *sorts* and *joins* are common processing chores. Search of fixed format data (e.g. indices or pointers) could make effective use of optical content-addressable memory which can be implemented by multiplexing a large number of holograms in a thick recording material like lithium niobate [Gaylord85]. The need for large amount of storage will probably be satisfied by using optical disks. Optical preprocessing of the retrieved data, without intermediate electrical conversion, may help deal with extreme data rates.

Currently, access times of optical disks are larger than those of magnetic disks. The reason is that the focusing optics are bulkier than the 'flying' miniature heads of magnetic disks. Data rates are comparable, with a promise for the future since the first are relatively new technology. However, in contrast with magnetic media, there are two promising possibilities for increased optical disk performance by at least two orders of magnitude both in terms of access time and sustained data rates. First, the read/write beam can deflected from track to track very rapidly (order of 100μs) by entirely optical means [Neff87]. Second, due to the non-interference of light beams and the relatively large head to medium spacing one

could imagine multiple beams being used for reading data with a single head carriage assembly [Carlin84] or an unfocused beam could simultaneously read data from more than one point of a transmissive disk surface [Mostafa87]. This coupled with the possibility of multiple heads will allow for enormous data rates. If one assumes that access times of a 100 μs and data rates of 200 Mbytes per second are achieved then this represents almost two orders of magnitude improvement over current magnetic disks. I/O systems will have to be designed with these rates in mind. Current electronics would be hard pressed. However, if data could be preprocessed "on the fly" in its optical form, then the ultimate data rate to the electronics would be much lower, and the data much "richer" in information. Intelligent use of optical pattern matching could provide us with a set of primitive operations that could help implement efficiently higher order functionality like, for instance, a subset of relational algebra operations.

The issues that are raised by the possibility of a different way of implementing digital computers are far from being just technological. One cannot afford to consider them in isolation and without questioning the applicability of current models for computing. This makes any attempt to pinpoint future directions risky and vulnerable to obsoleteness if revolutionary progress takes place. Electronics technology and the von Neumann model have been so well entrenched that straying away from these precepts was not common a few years ago. It also requires competence in a multitude of different scientific fields; it is only recently that we are witnessing such interdisciplinary efforts.

The field of digital optical computing is so young that nearly every kind of research would be beneficial. Optical devices, optical storage and optical

communications (not necessarily in that order of importance) need development before one can speak of general purpose optical computing.

Optical communications and storage are already setting the infrastructure on which more general applications of optics depend. New general directions seem to be worth exploring. Bridging the gap, or sharing ideas between the purely discrete, symbolic world of contemporary computer electronics and the inherently continuous and parallel modeling possible by optical phenomena is one. Dynamic "architecture compilation" driven by the algorithmic requirements of the problem being solved and the resources available is another. In the final analysis optics may have great potential in making computing even more pervasive. Continued efforts are necessary to determine if this potential is real.

References

[Abraham83] E. Abraham, C. T. Seaton and S. D. Smith, The Optical Digital Computer, Sci. Amer., pp 85-93, February 1983

[Bell86] Trudy E. Bell, Optical Computing: A field in flux, IEEE Spectrum, pp 34-57, August 1986

[Bloomberg85] D. S. Bloomberg and G. A. N. Connell, Prospects for Magneto-optic Recording, Compcon Proc., IEEE Spring 1985

[Carlin84] D.B. Carlin, J.P. Bednarz, C.J. Kaiser, J.C. Connolly, M.G. Harvey, Multichannel optical recording using monolithic arrays of diode lasers, Applied Optics, vol 23, no 22, pp 3994-4000, 14 November 1984

[Chen86] Peter Pin-Shan Chen, The compact disk ROM: How it works, IEEE Spectrum, April 1986, pp 44-49

[Fisher85] A. D. Fisher and C. L. Giles, Optical Adaptive Associative Computer Architectures, Compcon Proc., IEEE Spring 1985

[Hoagland85] Albert S. Hoagland, Information storage technology, A look at the future, IEEE Computer, July 1985, pp 60-67

[Huang84] Alan Huang, Architectural Considerations Involved in the Design of an Optical Digital Computer, Proc. of the IEEE, Vol. 72, no. 7, July 1984, pp 780-786

[Gaylord85] T.K. Gaylord, M.M. Mirsalehi, C.C. Guest, Optical digital truth-table look-up processing, Optical Engineering, vol 24, January/February 1985, pp 48-58

[Goodman84] J. Goodman, F. Leonberger. S.Y.Kung, R.A. Athale, Optical interconnections for VLSI systems, Proc. IEEE Vol. 72, July 1984, pp 850-866

[Goodman85] Joseph Goodman, Optical Interconnections: A Role for Optics in Computing of the Future, Compcon Proc., IEEE Spring 1985

[Ichioka84] Yoshiki Ichioka, Jun Tanida, Optical parallel logic gates using a shadow-casting system for optical digital computing, Proc. IEEE Vol. 72, no 7, July 1984, pp 787-801

[Jenkins85] B. K. Jenkins, Architectural Characteristics of Optical Logic Systems, Compcon Proc., IEEE Spring 1985

[Laub86] Leonard Laub, The evolution of mass storage, Byte, May 1986, pp 161-172

[Mnatsakanyan79] E. A. Mnatsakanyan, V.N. Morozov, and Yu. M. Popov Digital data processing in optoelectronic devices (review), Sov. J. Quantum Electron. 9(6), June 1979, pp 665-677

[Mostafa87] Yaser S. Abu-Mostafa, Demetri Psaltis, Optical Neural Computers, Sci. Amer., March 1987, pp 88-95

[Neches86] Philip M. Neches, The anatomy of a data base computer - revisited, Compcon Proc, IEEE Spring 1986

[Neff87] John, A. Neff, Major initiatives for optical computing, Optical Engineering, Vol 26, No 1, January 1987, pp 002-009

[Psaltis84] Demetri Psaltis, Two-Dimensional Optical Processing using One-dimensional Input Devices, Proc. of the IEEE, Vol. 72, No. 7, July 1984, pp 962-974

[Sawchuk84] Alexander A. Sawchuk and Timothy C. Strand, Digital Optical Computing, Proc. of the IEEE, Vol. 72, No. 7, July 1984, pp 758-779

[Smith81] Peter W. Smith and W. J. Tomlinson, Bistable Optical Devices promise subpicosecond switching, IEEE Spectrum. Vol. 8 , June 1981, pp 26-33

[West85] Lawrence West, GaAs Quantum Wells, PhD Dissertation, Lawrence Livermoore Labs. 1985

Appendix A-3

An Initial Design of a Very Large Knowledge Base Architecture

An Initial Design of a Very Large Knowledgebase Architecture

P. Bruce Berra
Department of Electrical & Computer Engineering
Syracuse University
Syracuse, New York 13244-1240
(315) 423-4445

January 1987

Abstract

In this paper we present an initial design for a very large knowledgebase architecture. The purpose of this architecture is to serve as a testbed for conducting research in very large data and knowledgebases. When completed the system will contain 100's of gigabytes of magnetic and optical disk storage, at least one half of a gigabyte of solid state memory, parallel paths with very wide bandwidths between elements, and multiple data and knowledge-base processors.

This work was supported by the Airforce Systems command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFb, DC 20332 under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

databases in the servicing of AI applications such as expert systems, but must also be extended and improved to process and manage knowledge as well. The future for both of these fields is very bright since much of what we now know as AI will be integrated with the database system. To understand this one need only consider the current flury of activity in the inter-face between logic programming and relational database, expert database systems and intelligent database systems. This marriage will allow AI researchers to address larger and more complex prob-lems and adds considerable vitality to database systems.

In this paper we discuss the initial design of a very large knowledgebase system architecture that we are building. The system will have considerable magnetic and optical disk capacity (100's of Gigabytes), wide bandwidth interconnections, large solid state memory (1/2 Gigabyte), special data and knowledge base processors and interfaces to several different architectures. The purpose of the system is to serve as a test bed in the study of very large data and knowledge base problems. To our knwoledge this system is unique in that we know of no other system of this magnitude that is specifically devoted to research in this field.

Very Large Knowledgebase Architectures

Shown in Figure 1 is an overall block diagram of the initial design of a very large knowledgebase architecture. At the left we show magnetic and optical disks. We invision that the system will have about 100 GBytes of magnetic disk and 500 GBytes of optical disk. Obviously, the control philosophies for accessing these two types of technologies will be different. Also, it is likely that read/write optical disks will be readily available in the next few years so that some mixture of read only and read/write will be used. Concentrating on magnetic disks for discussion purposes the data collector will receive data from

An Initial Design of a Very Large Knowledgebase Architecture

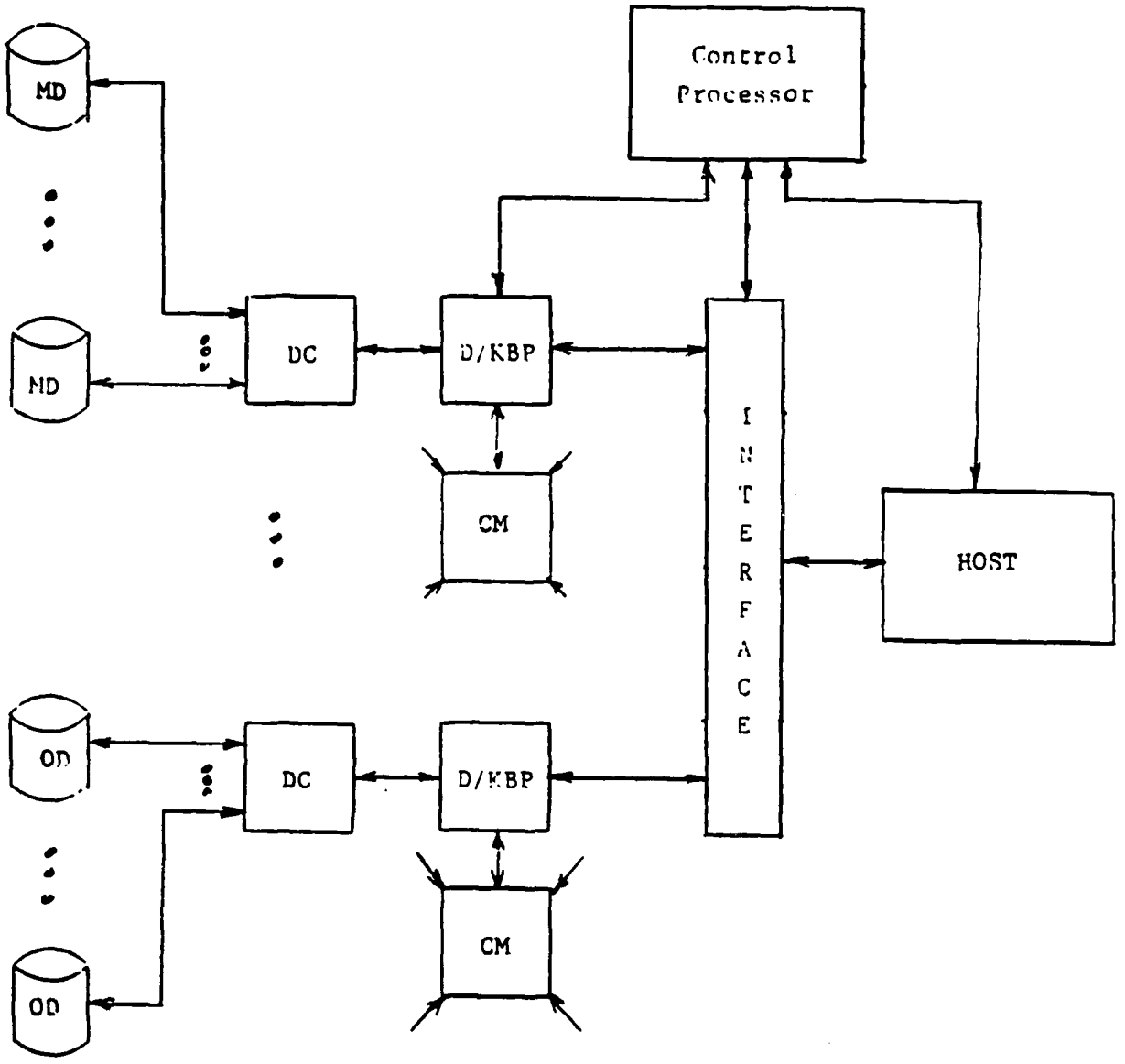
P. Bruce Berra

Introduction

Database management, as currently practiced, is a relatively mature field having had its origins in the 1960's. Databases affect our lives whether through weather predictions, airline reservations, stock quotations, food production, football players or unpaid parking tickets. In fact, the control and management of databases is a billion dollar industry that continues to grow and will have an even greater effect on our daily lives in the future. One of the major problems that has plagued us since the origin of the field is how to deal effectively with very large databases. To appreciate the magnitude of some databases one need only think about the hundreds of satellites that are beaming data back to the earth 24 hours/day, seven days a week or the vast information contained in libraries or the information on the millions of tax returns that must be entered into the Internal Revenue Service databases each year. Partially in response to the problems of large databases, researchers have been very active for the past 20 years in the development of database machines. These machines have as their primary focus the use of parallel processing to improve the performance of the database management function. Secondarily, they are concerned with the implementation of certain database functions in hardware rather than software thus improving performance.

In the last five to ten years there has been considerable interests in the use of database technology to increase the scope of artificial intelligence (AI) applications particularily in the knowledgebase area. This has placed new demands on the database field both in terms of increasing functionality and in terms of increased performance. In order to meet these demands we must develop new knowledgebase system architectures that are able to manage very large data and knowledgebases.

The new machines must not only be able to process the data from existing



MD - Magnetic Disks
OD - Optical Disks
DC - Data Collector
D/KBP - Data/Knowledge
CM - Common Memory

Figure 1. Very Large Knowledgebase Architecture

multiple disks simultane-ously. We envision that eight to ten disks will be controlled by a single data collector thus obtaining a data transfer rate of about 20-30 MBytes/sec. to each data/knowledgebase processor. The aggregate data rate could approach 500 MByte/sec. in the full system. Not shown in Figure 1 is the disk controller. We envision that each disk will have its own control processor and this proces-sor will share the controller function with the data collector.

The data and knowledgebase processor (D/KBP) acccepts data from the data collector at the aggregate rate of 20-30 MByte/sec. and either processes it or passes it through to the common memory. We will discuss passing data through to the common memory when we discuss a machine evaluation application. There are many D/KBP's each controlling eight to ten disks. Several D/KBP's share a common memory module. The aggregate capacity of all of the common memory modules is expected to be 1/2 GByte.

A more detailed block diagram of the D/KBP is shown in Figure 2. It performs two classes of operations on the data it controls. There are processes that respond to external commands from the control processor as shown in Figure 1 and internal processes that it must undertake to operate in a near optimal way. As previously mentioned we believe that much of the inferencing capabilities of current AI systems will become part of the database system to form an intelligent database or expert database system or perhaps the term knowledgebase system will take on that meaning in the future. We believe that new functions will be added to the database system to give it new functionality. For instance in work by Yokota and Itoh (1986) they discuss a relational knowledge base system that has the added functionality of unification-join and unification-restriction. The D/KBP will be designed to include appropriate addition capabilities. Our work as part of the Northeast Artifical Intelligence

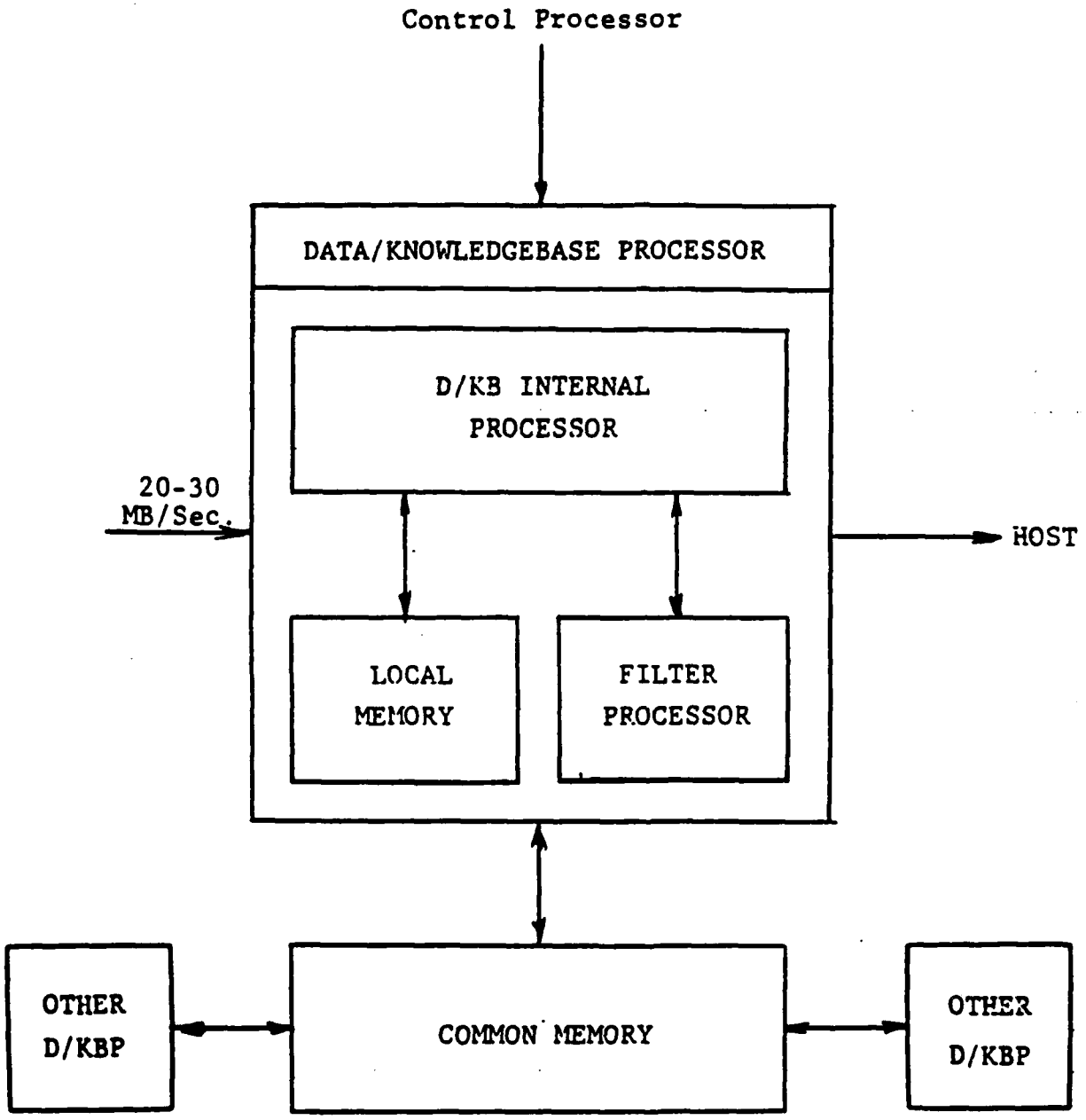


Figure 2. Data/Knowledgebase Processor

Consortium (Berra, et al 1987) will have a significant inpact on the design of the D/KBP.

With regard to internal optimization the D/KBP must be able to generate and control index information for the data it manages, it must be able to optimally place the data on disk for minimization of access and update time and it must be able to maintain the data in terms of security, integrity, backup and recovery. In addition the D/KBP contains a filter processor which performs the more common operations such as sort, merge, select, project and join.

Returning to Figure 1 the nonprocessed or processed data are placed in the common memory. These data will be removed via the control processor for some applications but mostly via the inter-face to the host. The bandwidth between the D/KBP and the common memory and between the common memory and the interface will be on the order of 100's of megabytes so as not to be a bottleneck.

Research Issues

We are concerned with the management of very large data and knowledgebases in the support of multiple inferencing mechanisms for logic programming. Since the evaluation of goals can require the accessing of the extensional database (EDB) of facts in very general ways one must often resort to indexing on all fields of the facts. Cast in relational database terminology each relation must be indexed and each attribute of each relation must also be indexed. For very large databases the amount of index data can become very large; in fact it may be as large as the data itself. Thus, if we have 500 Gbytes of fact data we can have 500 Gbytes of index data. We are currently trying to solve this problem (Berra, et al 1987) by using surrogate files. These are transformed representations of the index data with most of the information but with only 20% of the size. We have analyzed concatenated code words, superimposed code words

and transformed inverted lists as possible structures for the surrogate files. We will use concatenated code words (CCW) to illustrate some of the ideas. Perhaps the most simplified view of a CCW is that it is just a concatenation of the hashed values of the arguments in a tuple (in binary) with a unique identifier attached to it. The unique identifier is also attached to the untransformed tuple and is used as a pointer in the retrieval process. In order to access the database in response to a query from the logic programming inferencing mechanism we must first generate a query code word by transforming the known arguments to a binary representation. We must place them in their proper location vis-a-vis argument positions in the relation and fill the unknown argument positions with don't care values. This query code word is then used as the search argument in accessing the concatenated code word surrogate file. From this process we obtain a list of unique identifiers that are used to access the extensional database. The retrieved facts are then compared with the original query values to insure that hashing collisions have not occurred and the resulting facts are sent to the logic programming inferencing mechanism.

The use of surrogate files helps to improve retrieval performance because less processing is required due to its smaller size. However, additional performance can be obtained by distributing the surrogate file entries as uniformily as possible over all of the disks to allow for parallel processing. We are developing a special surrogate file processor that will be embedded in each of the D/KBP's. This device will utilize the query code word as a search argument to obtain the list of unique identifiers that qualify. Not only will this surrogate file processor be used for the process discussed above but certain relational operations (selection, join) can be performed on the CCW thus further improving performance.

As pointed out earlier we are concerned with the management of very large

data and knowledgebases in an multiple inferencing environment. However, the very large knowledgebase architecture will serve as a resource for many other interesting avenues of research. Among these are questions concerning the management of very large multimedia databases, the development of new embedded architectures, the comparative analysis of database indexing structures, the optimal reorganization of the database in response to usage and some of the more mundane questions concerned with concurrency control, back up, recovery and distribution. In the next section we discuss an additional use for the very large knowledgebase architecture, that of machine evaluation.

Machine Evaluation

One of the applications that is envisioned for the very large knowledgebase system architecture (KBSA) is the evaluation of experimental machines. In Figure 3 we indicates the configuration when the KBSA is being used for machine evaluation. When a machine is evaluated on the basis of performance (time per job, throughput, etc.) one must be able to keep the machine supplied with data. In fact, the date rate to the machine under test must be greater than the ability of the machine to handle it in order to obtain a realistic measure of performance. This requirement applies across the entire range of applications from processing intensive applications such as image processing to input/output intensive applications such as database management. To obtain a realistic measure of a machine's performance, both processing and input/output time must be taken into consideration. Thus, if all of the data will not fit into the machine's main memory, the time to input new data must be taken into account in the performance measure.

In order to illustrate some of these ideas and to describe some of the functions of the KBSA suppose that the machine under test has 64 MBytes of

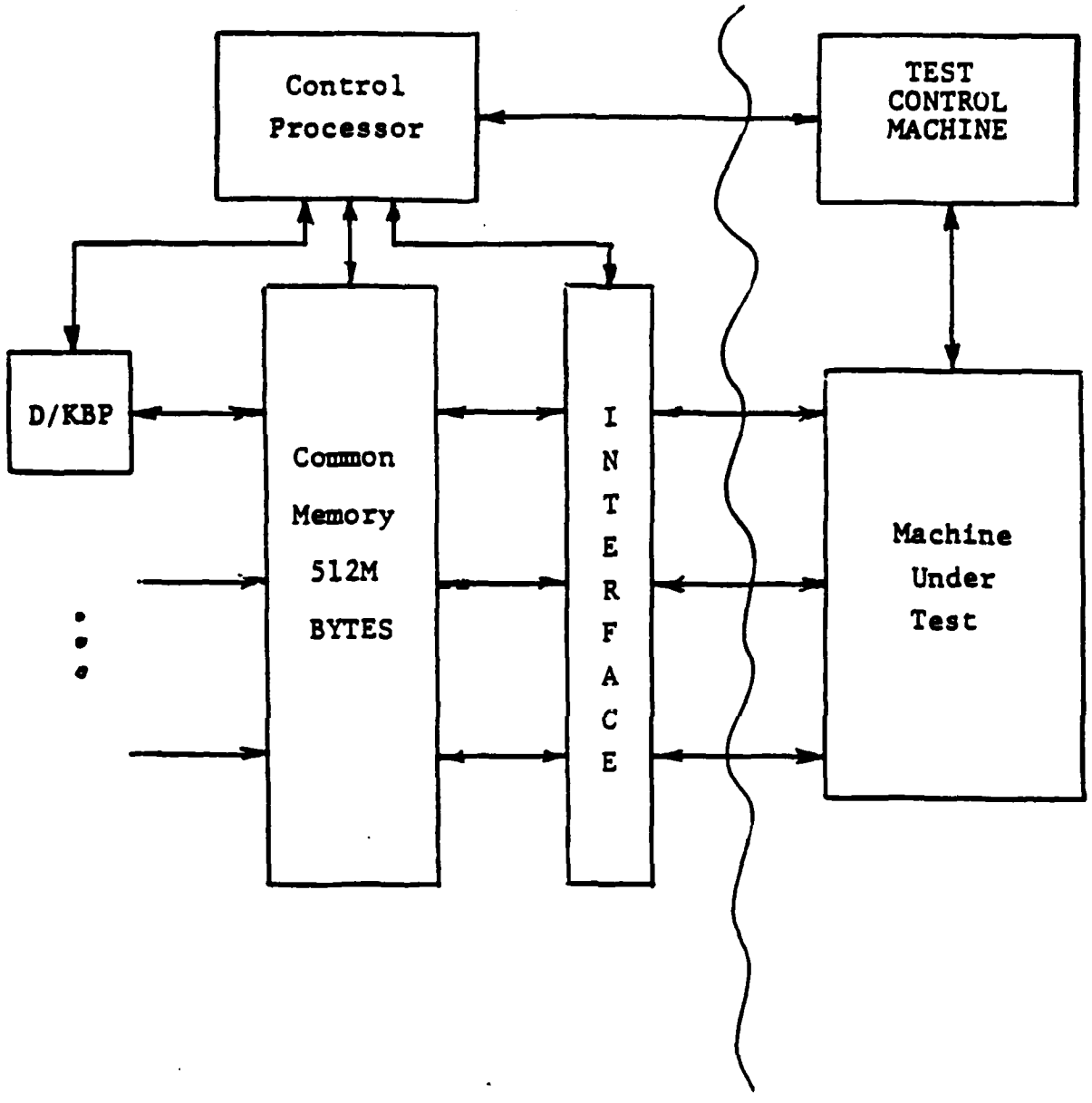


Figure 3. KBSA Configured for Machine Evaluation

memory which is equivalent to about 30,000 one to two page documents. After the first load of textual data in the 64 MBytes of memory has been processed, new data must be obtained from the KBSA. Thus, the load/process cycle will be carried out as many times as needed until the problem data have been exhausted. The time to complete the job then is the sum of the load and process time provided that they cannot be overlapped. In order to ensure a realistic test, the KBSA must have sufficient capacity and bandwidth to supply the data for a large problem at stress rates to the machine under test.

In problems that are processing intensive, the KBSA must also be able to supply data to the machine being evaluated at the highest rate it can handle. In the general case, suppose that the machine under test is input/output bound on problems of interest. Then, preprocessing can be performed in the KBSA in order to enrich the data being sent to the machine under test. Thus, in addition to testing the machine, the KBSA can identify some of the requirements for the secondary storage system to support the machine being evaluated.

The testing of machines places some severe constraints on the KBSA. It must be able to sustain output data rates in the hundreds of megabytes per second range and hava a data capacity in the hundreds of gigabytes. It must have the facility to provide raw data to a machine under test at stress data rates and must also be able to perform considerable processing activities in order to enrich the data being sent to the machine under test. It must be able to interface with a wide variety of machines from experimental machines under test to standard sequential machines. It must have some level of reconfigurability so that the above functions can be performed, and it must be partitionable so that it can interact with more than one machine.

Conclusion

In this paper we have attempted to describe a very large knowledgebase system architecture testbed and some of its possible attributes as a research tool. However, the design and development of the system is a research project in itself.

The overall design must be such that the generality is maintained in terms of reconfigurability, partitionability and intelligence while allowing for gradual growth of the system. That is, we envision that the system will transition through a series of design stages from essentially a disk farm to an intelligent processing system. We plan to do this gradually, building the system as we proceed. We are currently developing the surrogate file processor which will be part of and infulence the design of the D/KBP, we are conducting research aimed at the integration of AI and database, and we are designing the basic storage system (disks, controllers, and common memory). We expect the project to take three to five years and cost about five million dollars.

References

Berra, P.B., S.M. Chung, N.I. Hachem, "Computer Architecture for the Processing of a Surrogate File to a Very Large Data/Knowledgebase," to appear in IEEE Computer, March, 1987.

Yokota, H., H. Itoh, "A Model and an Architecture for a Relational Knowledge Base," The 13th Annual International Symposium on Computer Architecture, pp. 2-9, June 1986.

Appendix A-4

An Analysis of Surrogate File Structures for Very Large Knowledge Bases

An Analysis of Surrogate File Structures

for

Very Large Knowledge Bases

P. Bruce Berra
Soon Myoung Chung
Nabil I. Hachem

Dept. of Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13244-1240
(315) 423-4445

This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

ABSTRACT

This report presents techniques for managing a very large knowledge base to support multiple inference-mechanisms for logic programming. Because the evaluation of goals can require accessing data from the extensional data base (EDB) in very general ways, one must often resort to indexing on all fields of the extensional data base facts. This presents a formidable management problem in that the index data may be larger than the EDB itself. This problem becomes even more serious in the case of very large knowledge bases (100's of gigabytes) since considerably more hardware will be required to process and store the index data. In order to reduce the amount of index data considerably without losing generality, we form a surrogate file which is a hashing transformation of the facts. We discuss superimposed code words (SCW), concatenated code words (CCW) and transformed inverted lists (TIL) as possible structures for the surrogate file. We analyze them on the basis of storage space, time to answer queries and maintenance requirements of the surrogate files. Since these transformations are quite regular and compact we consider possible computer architectures for the processing of the surrogate file. We discuss the use of associative memory methods, as well as a back-end system that we are developing, in order to illustrate how nonsequential computer architectures can be used to advantage in solving this problem. Finally, we consider how one might perform relational operations on the surrogate file rather than on the full data.

Table of Contents

1. Introduction 1
2. Surrogate Files 6
3. System Model for SCW and CCW Techniques 12
3.1. Superimposed Code Word (SCW) 12
3.2. Concatenated Code Word (CCW) 14
4. Storage Requirement and Retrieval Performance of SCW and CCW Techniques 16
4.1. Size of SCW and CCW Surrogate Files 16
4.2. Query Response Time Using SCW and CCW Surrogate Files 18
4.2.1. Surrogate File Access Time 19
4.2.2. Surrogate File Searching Time 20
4.2.3. Database Access Time 20
4.2.4. Query Comparison Time 21
4.2.5. Query Response Time 21
5. Simulation and Analysis for SCW and CCW Techniques 22
5.1. Surrogate File Size 22
5.2. Query Response Time 29
5.3. Maintenance Aspects of SCW and CCW Surrogate Files 57
6. System Model for Transformed Inverted Lists 58
6.1. TIL1 Description 58
6.2. TIL2 Description 61
6.3. Partial Match on Multiple Argument Positions 63
7. TIL Storage Requirements 64
7.1. TIL1 Surrogate File Size 64
7.1.1. Secondary Index File Size 64
7.1.2. Primary Index File Size 65
7.2. TIL2 Surrogate File Size 66
7.2.1. Tertiary Index File Size 66
7.2.2. Secondary Index File Size 66
7.2.3. Primary Index File Size 67
8. Query Response Time of TIL 68
8.1. TIL1 Query Response Time 69
8.1.1. Primary Index Retrieval Time 69
8.1.2. Primary Index Search Time 70
8.1.3. Secondary Index Retrieval Time 70
8.1.4. Secondary Index Search Time 71

8.1.5. Intersection Time 71
8.2. TIL2 Query Response Time 72
8.2.1. Primary Index Retrieval Time 73
8.2.2. Primary Index Search Time 73
8.2.3. Secondary Index Retrieval Time 74
8.2.4. Secondary Index Search Time 74
8.2.5. Tertiary File Access Time 74
8.2.6. Intersection Time 74
8.3. Database Access Time (TIL1 or TIL2) 75
9. Simulation and Analysis of TIL Techniques 76
9.1. Transformed Inverted Lists Size 77
9.1.1. TIL1 Size 77
9.1.2. TIL2 Size 79
9.2. Transformed Inverted Lists Response Time 81
9.2.1 Analysis of TIL1 Time Equations 82
9.2.1.1. Surrogate Files Retrieval Time 82
9.2.1.2. Intersection Time 86
9.2.1.3. Database Access Time 91
9.2.1.4. TIL1 Query Response Time 93
9.2.2 Analysis of TIL2 Time Equations 105
9.3. Maintenance Aspects of TIL Surrogate Files 109
10. Comparison of Surrogate File Techniques 115
10.1. Size Comparison of SCW, CCW, TIL1 and TIL2 115
10.2. Query Response Time Comparison of SCW, CCW, TIL1 and TIL2 115
11. Computer Architecture for Surrogate File Processing 123
11.1. Associative Memory 123
11.2. Back End System 125
12. Relational Operations on the Surrogate File 129
13. Conclusion 131
Appendix 1: Average Number of Adjacent Blocks Containing the Same Argument Value 132
References 134

List of Figures

Figure 2.1 Partial Match Retrieval Using Superimposed Code Words (SCW) 8
Figure 3.1 Structure of SCW/CCW File 13
Figure 5.1 Effect of DB Size and the Average Number of Arguments in a Query on the SCW Surrogate File Size 23
Figure 5.2 Effect of False Drops on the SCW Surrogate File Size 25
Figure 5.3 Effect of the Average Number of Arguments in a Query on the SCW Surrogate File Size 26
Figure 5.4 Effect of the Average Redundancy on the CCW Surrogate File Size 27
Figure 5.5 SCW and CCW Surrogate File Size Comparison 28
Figure 5.6 Components of the SCW Query Response Time ($S_{db} = 10^5$ bytes) 31
Figure 5.7 Components of the SCW Query Response Time ($S_{db} = 10^7$ bytes) 32
Figure 5.8 Components of the SCW Query Response Time ($S_{db} = 10^9$ bytes) 33
Figure 5.9 Components of the CCW Query Response Time ($S_{db} = 10^5$ bytes) 34
Figure 5.10 Components of the CCW Query Response Time ($S_{db} = 10^7$ bytes) 35
Figure 5.11 Components of the CCW Query Response Time ($S_{db} = 10^9$ bytes) 36
Figure 5.12 Effect of the Number of Arguments in a Tuple on the SCW Query Response Time ($S_{db} = 10^5$ bytes) 37
Figure 5.13 Effect of the Number of Arguments in a Tuple on the SCW Query Response Time ($S_{db} = 10^7$ bytes) 38
Figure 5.14 Effect of the Number of Arguments in a Tuple on the SCW Query Response Time ($S_{db} = 10^9$ bytes) 39
Figure 5.15 Effect of the Number of Arguments in a Tuple on the CCW Query Response Time ($S_{db} = 10^5$ bytes) 40
Figure 5.16 Effect of the Number of Arguments in a Tuple on the CCW Query Response Time ($S_{db} = 10^7$ bytes) 41
Figure 5.17 Effect of the Number of Arguments in a Tuple on the CCW Query Response Time ($S_{db} = 10^9$ bytes) 42
Figure 5.18 Effect of False Drops and the Average Number of Arguments in a Query on the SCW Query Response Time ($S_{db} = 10^5$ bytes) 43
Figure 5.19 Effect of False Drops and the Average Number of Arguments in a Query on the SCW Query Response Time ($S_{db} = 10^7$ bytes) 44
Figure 5.20 Effect of False Drops and the Average Number of Arguments in

a Query on the SCW Query Response Time ($S_{db} = 10^9$ bytes) 45
Figure 5.21 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the SCW Query Response Time ($S_{db} = 10^5$ bytes) 47
Figure 5.22 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the SCW Query Response Time ($S_{db} = 10^7$ bytes) 48
Figure 5.23 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the SCW Query Response Time ($S_{db} = 10^9$ bytes) 49
Figure 5.24 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the CCW Query Response Time ($S_{db} = 10^5$ bytes) 50
Figure 5.25 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the CCW Query Response Time ($S_{db} = 10^7$ bytes) 51
Figure 5.26 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the CCW Query Response Time ($S_{db} = 10^9$ bytes) 52
Figure 5.27 SCW and CCW Query Response Time Comparison ($S_{db} = 10^5$ bytes) 54
Figure 5.28 SCW and CCW Query Response Time Comparison ($S_{db} = 10^7$ bytes) 55
Figure 5.29 SCW and CCW Query Response Time Comparison ($S_{db} = 10^9$ bytes) 56
Figure 6.1 A Simple Knowledge Base Relation 59
Figure 6.2 TIL1 for Ar_2 in Figure 6.1 60
Figure 6.3 TIL2 for Ar_2 in Figure 6.1 62
Figure 9.1 Effect of the DB Size and the Number of Arguments in a Tuple on the TIL1 Surrogate File Size 78
Figure 9.2 Effect of the DB Size and the Number of Arguments in a Tuple on the TIL2 Surrogate File Size 80
Figure 9.3 Effect of the Number of Arguments in a Query on TIL1 Index Retrieval Time ($S_{db} = 10^5$ bytes) 83
Figure 9.4 Effect of the Number of Arguments in a Query on TIL1 Index Retrieval Time ($S_{db} = 10^7$ bytes) 84
Figure 9.5 Effect of the Number of Arguments in a Query on TIL1 Index Retrieval Time ($S_{db} = 10^9$ bytes) 85
Figure 9.6 Effect of the Number of Arguments in a Query on TIL Intersection Time ($S_{db} = 10^5$ bytes) 87
Figure 9.7 Comparison of TIL Intersection Time for Low Values of the Average Redundancy and Medium to Large DB Sizes 88
Figure 9.8 Effect of the Number of Arguments in a Query on TIL Intersection Time ($S_{db} = 10^7$ bytes) 89
Figure 9.9 Effect of the Number of Arguments in a Query on TIL Intersection Time ($S_{db} = 10^9$ bytes) 90
Figure 9.10 TIL Database Access Time ($S_{db} = 10^5$ bytes) 92

Figure 9.11 TIL Database Access Time ($S_{db} = 10^7$ bytes) 94
Figure 9.12 TIL Database Access Time ($S_{db} = 10^9$ bytes) 95
Figure 9.13 TIL Database Access Time for Low Values of the Average Redundancy and Medium to Large DB Sizes 96
Figure 9.14 Components of the TIL1 Query Response Time ($S_{db} = 10^5$ bytes, $R_q = 1$) 97
Figure 9.15 Components of the TIL1 Query Response Time ($S_{db} = 10^5$ bytes, $A_r = 6$, $R_q = 2$) 99
Figure 9.16 Components of the TIL1 Query Response Time ($S_{db} = 10^5$ bytes, $A_r = 6$, $R_q = 4$) 100
Figure 9.17 TIL1 Query Response Time for Medium and Large DB Sizes ($S_{db} = 10^7$ and 10^9 bytes, $R_q = 1$) 101
Figure 9.18 TIL1 Query Response Time for Medium and Large DB Sizes ($S_{db} = 10^7$ and 10^9 bytes, $A_r = 2$, $R_q = 2$) 102
Figure 9.19 Components of the TIL1 Query Response Time ($S_{db} = 10^7$ bytes, $A_r = 6$, $R_q = 4$) 103
Figure 9.20 Components of the TIL1 Query Response Time ($S_{db} = 10^9$ bytes, $A_r = 6$, $R_q = 4$) 104
Figure 9.21 Effect of the Number of Arguments in a Query on TIL2 Index Retrieval Time ($S_{db} = 10^5$ bytes) 106
Figure 9.22 Effect of the Number of Arguments in a Query on TIL2 Index Retrieval Time ($S_{db} = 10^7$ bytes) 107
Figure 9.23 Effect of the Number of Arguments in a Query on TIL2 Index Retrieval Time ($S_{db} = 10^9$ bytes) 108
Figure 9.24 TIL2 Query Response Time ($S_{db} = 10^5$ bytes, $R_q = 1$) 110
Figure 9.25 TIL2 Query Response Time ($S_{db} = 10^7$ and 10^9 bytes, $R_q = 1$) 111
Figure 9.26 TIL2 Query Response Time ($S_{db} = 10^5$ bytes, $A_r = 6$, $R_q = 4$) 112
Figure 9.27 TIL2 Query Response Time ($S_{db} = 10^7$ and 10^9 bytes, $A_r = 6$, $R_q = 4$) 113
Figure 10.1 Surrogate File Size Comparison ($S_{db} = 10^5$ bytes) 116
Figure 10.2 Surrogate File Size Comparison ($S_{db} = 10^7$ bytes) 117
Figure 10.3 Surrogate File Size Comparison ($S_{db} = 10^9$ bytes) 118
Figure 10.4 Query Response Time Comparison ($S_{db} = 10^5$ bytes) 119
Figure 10.5 Query Response Time Comparison ($S_{db} = 10^7$ bytes) 121
Figure 10.6 Query Response Time Comparison ($S_{db} = 10^9$ bytes) 122
Figure 11.1.a SCW Mapped into Associative Memory 124
Figure 11.1.b CCW Mapped into Associative Memory 124
Figure 11.1.c TIL Mapped into Associative Memory 124
Figure 11.2 Back End System for Fact Management 126

List of Tables

Table 4.1 Summary of Notations 17
Table 5.1 The Values of Parameters Used in the Simulation 30

1. Introduction

One of the most rapidly growing fields of artificial intelligence is expert systems. These systems are composed of a knowledge base that consists of rules and facts and an inference mechanism that can be utilized to respond to queries posed by users. The objective of such systems is to capture the knowledge of the best experts in particular domains and make it generally available to nonexpert users. The current state of the art of such systems is that they focus on narrow domains, have small knowledge bases and are thus limited in their application.

As these systems are expanded and more widely used, increased demands will be placed on the management of their knowledge bases. The intensional database of rules will become large and present a formidable management task in itself. But, the major management activity will be in the access, update and control of the extensional database of facts. The volume of facts will be in the gigabyte range, and we can expect to have general extensional databases that serve multiple inference mechanisms that may be logic, frame, production rules or semantic network oriented.

Our current research focuses on the development of special computer architectures to access, update and control very large knowledge bases that support multiple logic programming inferencing mechanisms. The details of logic programming are widely known and will not be discussed here (Clocksin and Mellish 1984). However, we need to provide a simple example in order to set the stage for the very large knowledge bases problem that we are addressing in our research.

As an example consider the following simple logic programming problem.

1. grandfather(X,Y) ← father(X,Z), parent(Z,Y)
2. parent(X,Y) ← father(X,Y)
3. parent(X,Y) ← mother(X,Y)
4. father(pat, tiffany) ←

   father(don, louise) ←

   .

   .

   .

5. mother(mary, louise) ←

   mother(lisa, tiffany) ←

   .

   .

   .

6. ← grandfather(X, tiffany)

The first three clauses form the intensional database of rules for this problem, the next two sets form the extensional database of facts regarding fathers and mothers and the last statement is the goal (or query). The solution of the problem (the satisfaction of the goal) consists of the names of the grandfathers of Tiffany. In the solution of this problem the searching of the father and mother facts will take place on the second argument-position in order to determine values for the first argument position that can be used for further processing. Thus, we need to find Tiffany's mother and father before finding her grandfathers. If we ask a similar but slightly different query

← grandfather(ray, X)

searching of the extensional database takes place entirely on the first argument of the father and mother facts. In this case we are trying to determine Ray's grandchildren.

In order to generalize consider the fact type (relation) shown below

r(a, b, c, ..., n).

It is clear from the example given above that in the general case one would want to access a relation on the basis of any subset of the arguments of the relation depending upon how the query was stated. This problem then becomes a special case of the partial match retrieval problem. In this case we are given a specific relation name and values for a subset of the arguments and desire all tuples that satisfy the search criteria. The desired tuples would then be returned to the inferencing mechanism for further processing.

In the context of very large knowledge bases the question arises as to how to obtain the desired facts in the minimum amount of time. If we assume that any subset of the attributes of a relation is equally likely to form the basis for a query then we will probably have no choice other than to index on all attributes in order to insure reasonable performance. However, if we have 500 gigabytes of fact data we could have 500 or more gigabytes of index data. This is an enormous amount of index data to deal with.

We are currently taking two approaches to the solution of this problem. The first is to develop special hardware to process surrogate files[1] that can be

[1] The term surrogate file dates back to early work in information retrieval. Other terms such as signature files and descriptor files have also been used to describe similar structures. See Faloutsos 1985 for a review of various methods for accessing textual data.

used to efficiently access the extensional database and the second is to consider optical techniques that can potentially increase input/output rates by orders of magnitude and thus greatly reduce access time to the extensional database. This report concerns the first approach.

As was previously pointed out index information for accessing the extensional database may be as large as or larger than the facts themselves. In the surrogate file approach we construct a transformed representation of the index data which is then processed to gain access to the actual data. This collection of transformed data is called the surrogate file. The transformation can be performed with well chosen hashing functions and yield a surrogate file that is much smaller than the original data. We have analyzed superimposed code words (SCW), concatenated code words (CCW), transformed inverted lists (TIL) and combinations of these as basic structures of the surrogate file.

This report concerns the results of our analysis. In the next section we discuss the surrogate file approach and introduce the technique of SCW, CCW, and TIL as possible ways to structure the surrogate file. We then consider each of these techniques in detail. We analyze them on the basis of storage space required for the surrogate file, time to answer queries and updating of the surrogate file. In the timing analysis we take into consideration the accessing of the surrogate file, the processing of the surrogate file, the accessing of the extensional database, and any processing that may be required in the process. Initially we treat each of the techniques separately and then combine the results for comparative purposes.

In the last section we consider two possible general approaches to the development of computer architecture that can be used to manage the extensional database. These include an associative memory approach and a back end

system approach. Finally, we close with some thought on the performance of relational operations on the surrogate files rather than the extensional database.

2. Surrogate Files

We assume that many different relations (fact types) with varying cardinalities exist in the very large extensional database that we are considering. Furthermore, we assume that the tuples are stored in such a way that one first accesses the relation followed by an access to a particular tuple via its unique identifier. We will obtain the name of the relation and a subset of values along with their positions in the relation from the logic programming process when it requests data. Thus, the storage structure for the actual facts themselves would be very simple and a method such as extendible hashing (Fagin 1979) could be used to guarantee retrieval of a given fact in at most two disk accesses. This presupposes that all secondary key retrievals will take place on the surrogate file or through post processing of the retrieved tuples if there are many different types of users of the same database.

The principal techniques that we have considered for the construction of the surrogate file include concatenated code words (CCW), superimposed code words (SCW), combinations of CCW and SCW, and transformed inverted lists (TIL). We will use CCW and SCW to illustrate the ideas.

Suppose we have a fact type called BORDERS which is given as follows:

BORDERS (Country 1, Country 2, Body of Water).

For a particular instance

BORDERS (France, Italy, Mediterranean).

we would first hash the individual values,

H(France)	H(Italy)	H(Mediterranean)
↓	↓	↓
100...01	010...00	110...00

concatenate them, and then attach a unique identifier to obtain the CCW

100...01|010...00|110...00|00...01

where the vertical line shows the boundaries. The SCW (Roberts 1979) would be formed as follows

H(France)	→	100...01
H(Italy)	→	010...00
H(Mediterranean)	→	110...00
		110...01|00...01

with the binary representations logically ORed together. The unique identifier is attached as shown.

The unique identifier would also be added to the fact itself. Thus, a surrogate file would exist with an entry for each fact in the database and the individual entries would be linked to the facts by their unique identifiers.

As illustrated in Figure 2.1 using SCW's if one wanted to retrieve a fact based upon one or more arguments, one would pass each of the arguments through the same hashing function, and OR their fixed length binary representations together in order to generate the query code word (QCW). We would then compare the QCW with all of the SCW for that fact type seeking matches on the

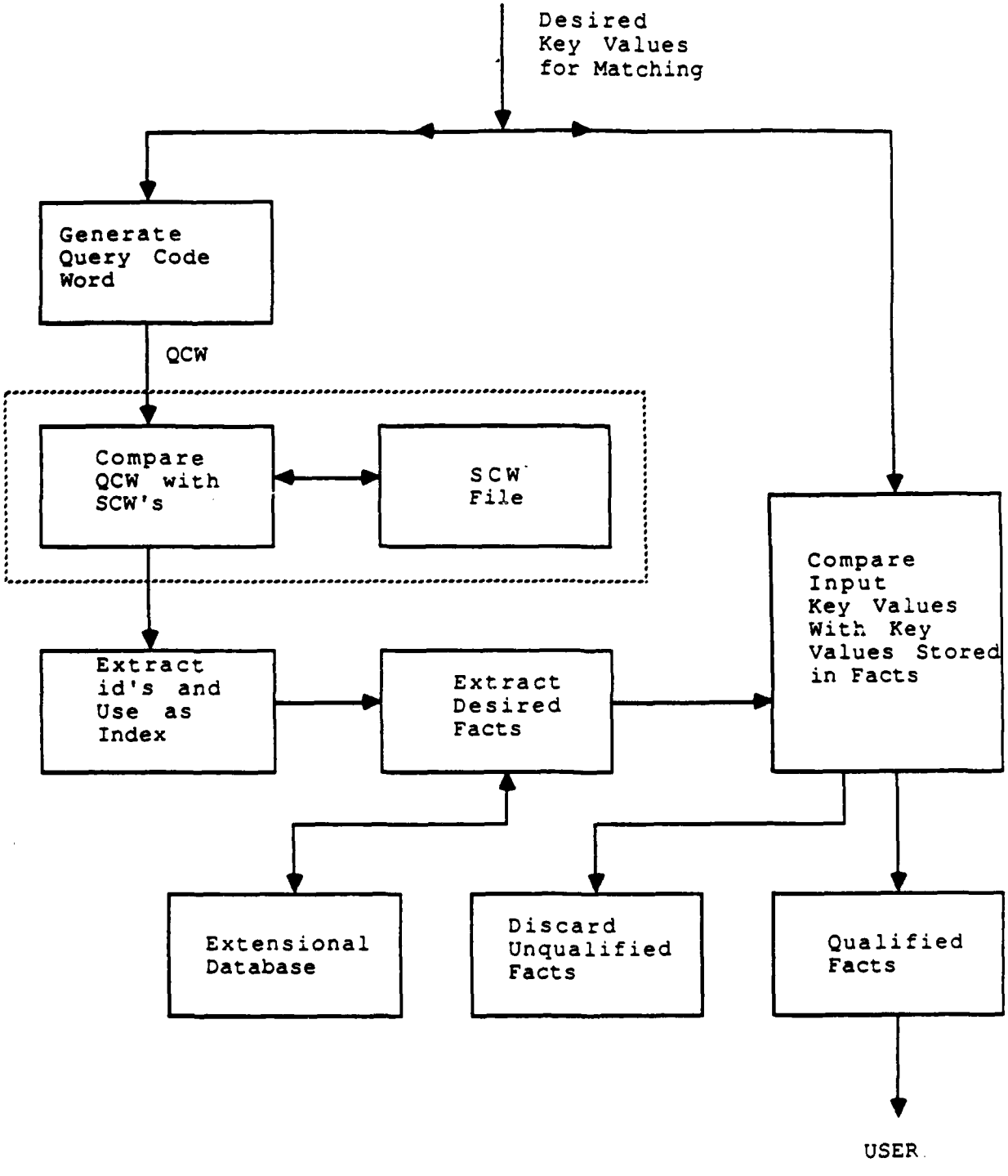


Figure 2.1 Partial Match Retrieval Using Superimposed Code Words (SCW).

ones of the QCW. All of the SCW's that had ones in the same positions as the QCW would be possible qualifiers. The id's of the possible qualifiers would then be used to retrieve the facts in question. The input key values would then be compared with the respective key values of the retrieved facts in order to determine possible matches. If the desired fact(s) exists within the database it will be found. However, a number of facts may be retrieved that do not match the input key values, called false drops. The use of SCW's in the context of logic programming has also been discussed by Wise and Powers (1984).

The concatenation of the code words is one extreme while the method of SCW's is the other. The concatenation method almost insures (subject to the quality of the hashing function in avoiding collisions) that the unique identifier(s) that is obtained is indeed the one(s) that is desired. However, it does so at the expense of a longer word length. The method of SCW reduces this longer word length but does so at the expense of increasing the number of pages retrieved that do not contain the desired facts.

In order to fix the ideas consider a three argument relation taken from a much larger extensional database. We will assume that the tuples consist of only a M byte of data and thus comprise approximately 15,000 facts. To develop a concatenated code word with unique identifier for each fact would require about 20 bits for each argument position and 14 bits for the unique identifier. Each tuple of a relation must have a unique identifier if it is to adhere to the basic assumptions of the relational model; if it does not exist within the tuple it will have to be concatenated to it. We will assume that it must be added. Thus the total number of bytes would then be 140 KB or 14 percent of the total. With a SCW approach these values can be reduced to 42 bits per fact or about 8 percent of the total. This number is obtained by allowing 14 bits for the unique identifier

and 28 bits for the SCW. Roberts (1979) suggests that the length of the SCW should be such that the number of ones and zeros occur with equal probability in order to minimize the false drops.

It may be that there is a point in between these two methods that is optimal. For instance, Lloyd et al (1980, 1982) has taken an interesting approach in which he selects bit values from various positions of the binary representation of the argument values in the facts and concatenates them to form a code word. Also, we have explored the development of a method that is a hybrid of the SCW and the CCW method. The resulting code word has unique portions for each argument and non unique portions that represent the ORing of two or more arguments. While this method reduces the length of the code word it does however, increase the probability of false drops from that of CCW.

Another approach that we have investigated is the use of transformed inverted lists. In this method we hash the original argument values for each position and keep a list of unique identifiers for the original facts. If one is searching for a fact based on a single argument this method has definite advantages since less index data needs to be processed. However, when a number of arguments are involved, additional processing will be required since lists will have to be intersected.

In this report we have assumed ideal hashing functions. However, we realize that the selection of the proper function is critical and full analysis is required in order to insure that the number of false drops is kept to a minimum.

With regard to maintenance, a varying amount of work is required depending upon whether SCW, CCW or TIL structures are being used. In the case of adding a new tuple to a relation the SCW or CCW would have to be generated,

the unique identifier attached to the tuple and the code word, the tuple stored in the EDB and the code word appended to the end of code word subfile for that relation. In the case of TIL structured files each of the lists of that relation would have to be processed either by adding a unique identifier to the list of current unique identifiers for that value or adding a new value and unique identifier to the list.

In the case of deletion of a tuple, the entry in the surrogate file would also have to be found and deleted when SCW and CCW are being used. If TIL are being used, an entry in each list must be found and then deleted. Considering changing the value content of a field, a new code word must be generated and the old one deleted when SCW's are being used. When CCW's are being used the change need only be made to the portion of the code word in question. With TIL files only the entry in the list being changed need be processed.

In this section we have attempted to provide a non mathematical description of the various approaches to the structure of the surrogate file. In the next section we deal with SCW and CCW since they are similar. We develop equations for the size of the surrogate file and for the query response time for facts. We then use those equations in a simulation in order to obtain numerical data on the performance of these structures. We then consider two forms of the TIL structure TIL1 and TIL2 which are similar but have some important differences. Again we develop equations for these structures considering size and query response time that are used in obtaining simulation data. We then make comparisons among all techniques.

3. System Model for SCW and CCW Techniques

In this section we develop models for the SCW and CCW techniques. We repeat some of the ideas that were introduced earlier for completeness and the development of notation for the model.

3.1. Superimposed Code Word (SCW)

Let a record D contain A_r argument values, $D=\{d_1,d_2,\cdots,d_{A_r}\}$. Each argument value $(d_i, 1 \leq i \leq A_r)$ can be mapped into a binary representation by a hashing function. The binary representation can be converted to another binary representation with pre-defined length and pre-defined weight, called a binary code word (BCW), by using a pseudo random number generator. The weight of a binary representation is the number of 1's in the binary representation. The process of generating a BCW from an argument value is well described in Roberts(1979). The SCW of a record is generated by ORing A_r BCWs obtained from A_r argument values and then attaching the unique identifier of the record which is the pointer to the database or can be converted to the actual pointer to the database. Figure 3.1 shows the structure of the SCW file which contains all the SCWs for records.

With the structure in Figure 3.1, the retrieval process as indicated in Figure 2.1 is as follows:

1) Given a query, obtain a query code word (QCW) by ORing BCWs corresponding to argument values in the query.

2) Obtain a list of unique identifiers to all records whose SCWs satisfies

QCW=QCW .AND. SCW

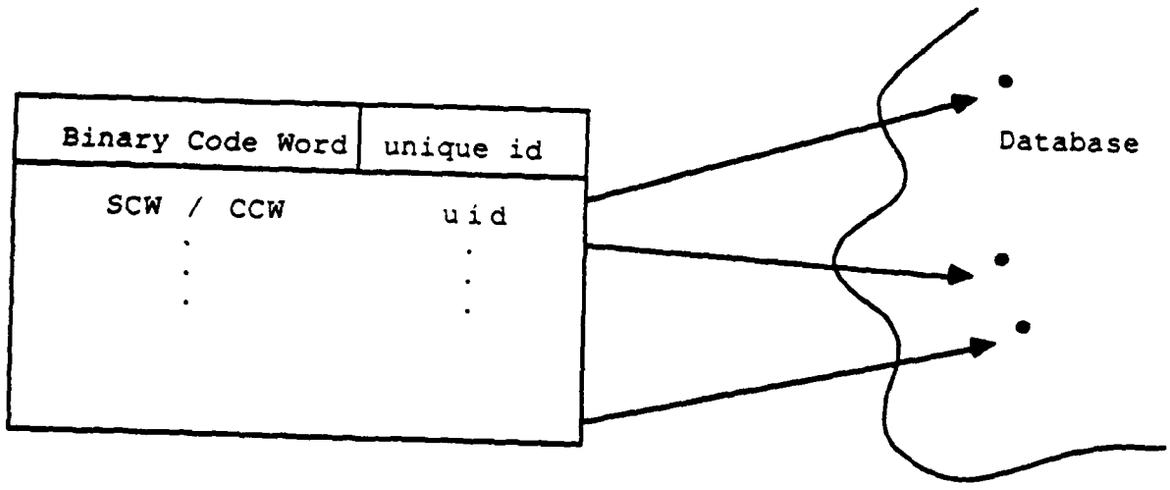


Figure 3.1 Structure of SCW/CCW File

that is, obtain a list of all SCW's that have 1's in the same position as the QCW by sequentially ANDing the QCW with all entries in the SCW file.

3) Retrieve all records pointed to by the unique identifiers obtained in step 2 and discard the records not satisfying the query. These are called "false drops". The records satisfying the query are called "good drops". The false drops are caused by ORing BCWs which make records with different argument values have the same SCWs.

4) Return the good drops to the user.

3.2. Concatenated Code Word (CCW)

The CCW of a record is generated by simply concatenating the binary representations of all argument values and attaching the unique identifier of the record. With this technique the binary representations (BRs) of all argument values have the same length.

The retrieval process with CCW file is as follows:

1) Given a query, obtain a query code word (QCW) by concatenating BRs corresponding to argument values specified in the query. The portion of the query code word for argument values which is not specified in the query is filled with don't care symbols.

2) Obtain a list of unique identifiers to all records whose CCWs satisfies

$$QCW=CCW$$

by sequentially comparing the QCW with all CCWs in the CCW file.

3) Retrieve all records pointed to by the unique identifiers obtained in step 2

and compare the corresponding argument values of the records with the query values to insure that collisions have not occurred. Any collisions are equivalent to false drops as discussed above. However, in this report we assume that no collisions have occurred.

4) Return the good drops to the user.

4. Storage Requirement and Retrieval Performance of SCW and CCW Techniques

Storage requirements can be expressed by the size of surrogate files and retrieval performance can be measured by the query response time for a given query. Notations that are frequently used in this report are shown in Table 4.1.

4.1. Size of SCW and CCW Surrogate Files

The equations for the size of the SCW and CCW files are obtained in this section under the assumption, that, if the input values are different from each other, the selected hashing function maps those values into different output values, that is, there are no collisions by the hashing function.

With the above assumption, Roberts(1979) presented the optimal bit length of a BCW in a SCW in terms of the number of arguments in a record (A_r), the average number of arguments in a query (R_q), the number of facts in the data-base (N), and the average number of false drops (DF). The equation for the optimal bit length (b_{bcw}) is given as

$$b_{bcw} = \left\lceil \frac{A_r}{R_q} \frac{[\ln(N)-\ln(DF)]}{[\ln(2)]^2} \right\rceil \tag{4.1}$$

The SCW also contains its unique identifier which must be greater than or equal to $\log_2 N$, thus the minimal bit length of a SCW (b_{scw}) is

$$b_{scw} = (b_{bcw} + \lceil \log_2 N \rceil) \tag{4.2}$$

Hence, the minimal size of the SCW file (S_{scw}) is as follows.

$$S_{scw} = b_{scw} \times N \tag{4.3}$$

Notations	Meanings
A_r	Number of arguments in a tuple
R_q	Average number of arguments in a query
DG	Average number of good drops
DF	Average number of false drops
S_{db}	Size of the database in bytes
N	Number of tuples in the database
S	Size of surrogate file in bits
B	Size of a block in bytes
BR	Binary representation
BCW	Binary code word
b_{bcw}	bit length of a binary code word
QT	Query Response time
IA	Time for retrieving surrogate files
IC	Time for comparing a query code word with code words in surrogate files
DA	Time for retrieving tuples in the database
DC	Time for comparing a query with tuples retrieved from the database
IT	TIL intersection time
C_i	Value distribution factor, that is, the average number of tuples which have the same value in the i-th argument
C_g	Average of value distribution factor (Average redundancy)

Table 4.1 Summary of Notations

For a CCW file, the minimal size can be derived from the fact that the bit length of the hashing function output for an argument in a record must be at least $\left\lceil \log_2 \frac{N}{C_i} \right\rceil$ where C_i, called the value distribution factor, is the average number of records whose i-th arguments have the same value. From this fact, we can derive the minimal sizes of the CCW file (S_{ccw}).

A CCW contains the binary representation of each argument value and its unique identifier. Hence, the minimal size of CCW file is

$$S_{ccw} = \left(\sum_{i=1}^{A_r} \left\lceil \log_2 \frac{N}{C_i} \right\rceil + \left\lceil \log_2 N \right\rceil\right) \times N \qquad (4.4)$$

4.2. Query Response Time Using SCW and CCW Surrogate Files

The query response time depends largely on the size of surrogate files and the method of obtaining pointers from the surrogate files. For the size of surrogate files, the equations derived in the previous section are used. Also, it is assumed that sequential search is performed on the SCW or CCW files.

In general, the retrieval process using surrogate files can be divided into several sub-processes as follows:

1. Access to the surrogate files to read the code words from those files.
2. Comparing the QCW of a query with code words and obtaining a list of pointers (unique id's) to the database.
3. Access to the database to read the records pointed to by the pointers obtained in 2.

4. Comparing the query with the records retrieved from the database. This step is to discard the false drops. Therefore, this step is necessary only for SCW technique, or if collisions are considered in the CCW case.

4.2.1. Surrogate File Access Time

Let T_b be the average time for access (seeking and reading) to a block in the surrogate files, and let each block in the surrogate files be randomly distributed over the storage media. Then,

$$T_b = \text{Average seek time} + \frac{B}{\text{Transfer rate}} \tag{4.5}$$

With SCW or CCW files, sequential search is assumed. Therefore, the total time for access to SCW or CCW files (IA_{scw} or IA_{ccw}) can be easily derived as

$$IA_{scw} \text{ (or } IA_{ccw}) = T_b \times \left\lceil \frac{S_{scw} \text{ (or } S_{ccw})}{B} \right\rceil \tag{4.6}$$

If the blocks in the surrogate files reside consecutively in a storage, then the time for access to SCW or CCW files will be considerably reduced because the disk position seeking is required only once per cylinder on the disk, that is,

$$IA_{scw} \text{ (or } IA_{ccw}) = \frac{B}{\text{Transfer rate}} \times \left\lceil \frac{S_{scw} \text{ (or } S_{ccw})}{B} \right\rceil \tag{4.7}$$

$$+ \text{ Average seek time} + \text{Rotational delay}$$

$$+ (\#\text{ of Cylinders for } S_{scw} \text{ (or } S_{ccw}) - 1) \times$$

$$(\text{Minimum seek time} + \text{Rotational delay})$$

4.2.2. Surrogate File Searching Time

With SCW or CCW files, all the code words are compared with the query code word (QCW) of the query. Therefore, the total time of searching the SCW (IC_{scw}) or CCW file (IC_{ccw}) is

$$IC_{scw} \text{ (or } IC_{ccw}) = T_c \times N \tag{4.8}$$

where T_c is the average time for comparing the QCW of a query with a code word and is given as

T_c = the number of bytes in a QCW × time for byte comparison / 2.

4.2.3. Database Access Time

The average number of accesses to the database is the average number of good drops. However with the SCW file, the average number of false drops must also be considered. Also, the records to be retrieved are assumed to be randomly distributed over the database. The time for access to the database for each technique is as follows:

$$DA_{scw} = T_{bd} \times \left\lceil \frac{S_{db}}{B} \right\rceil \times \left(1-\left(1-\frac{1}{\left\lceil \frac{S_{db}}{B} \right\rceil}\right)^{DG+DF}\right) \tag{4.9}$$

$$DA_{ccw} = T_{bd} \times \left\lceil \frac{S_{db}}{B} \right\rceil \times \left(1-\left(1-\frac{1}{\left\lceil \frac{S_{db}}{B} \right\rceil}\right)^{DG}\right)$$

where T_{bd} is the average time of retrieving a block in the database, DG the average number of good drops, and DF the average number of false drops with SCW

technique. In this report, DG is approximated by using C_i and N.

$$DG = \begin{cases} N \prod_{i \in R_q} \left(\frac{C_i}{N}\right) & \text{if } DG > 1 \\ 1 & \text{otherwise} \end{cases}$$

4.2.4. Query Comparison Time

Let T_{cd} be the time for comparing the query with a record. Then, the time spent for comparing retrieved records with the query in the SCW technique (DC_{scw}) is

$$DC_{scw} = T_{cd} \times (DF + DG) \tag{4.10}$$

Since we assumed that hashing function has no collision, each CCW corresponds to just one record in database. Therefore, comparison between query and retrieved record is not necessary in the CCW case.

4.2.5. Query Response Time

The query response time for a given query is the summation of all the sub-process time.

$$QT_{scw} = IA_{scw} + IC_{scw} + DA_{scw} + DC_{scw} \tag{4.11}$$

$$QT_{ccw} = IA_{ccw} + IC_{ccw} + DA_{ccw}$$

5. Simulation and Analysis for SCW and CCW Techniques

Simulations are performed with the equations for the size of surrogate files and the query response time using CCW and SCW techniques assuming that the surrogate files are consecutively stored in storage media and there is no free space in the surrogate files.

5.1. Surrogate File Size

For the simulation of the surrogate file size , it is assumed that the database remains at the same size regardless of variation of A_r and 15 bytes are used for each argument values. Therefore, N, the number of records in database, can be calculated as follows:

$$N = \left\lfloor \frac{S_{db}}{15 \times A_r} \right\rfloor$$

where S_{db} represents the actual database size not including the unique identifiers for each tuple of database. We also assumed that each argument of a tuple in the database has the same redundancy value, C_g, which is the average of the C_i's:

$$C_g = \frac{\sum_{i \in A_r} C_i}{A_r}.$$

The results for the size simulation are shown in Figures 5.1 through 5.5. In Figure 5.1 we plot the size of the SCW surrogate file (S_{scw}) as a function of the number of arguments (A_r) in a tuple. The size of the surrogate file is expressed as a percentage of the database. The database sizes are 10^5, 10^7 , and 10^9 bytes while the average number of arguments in a query (R_q) takes on the values one and two. Note that S_{scw} increases with the size of the database (S_{db}) but decreases with R_q. The reasons for this behavior are readily apparent from equa-

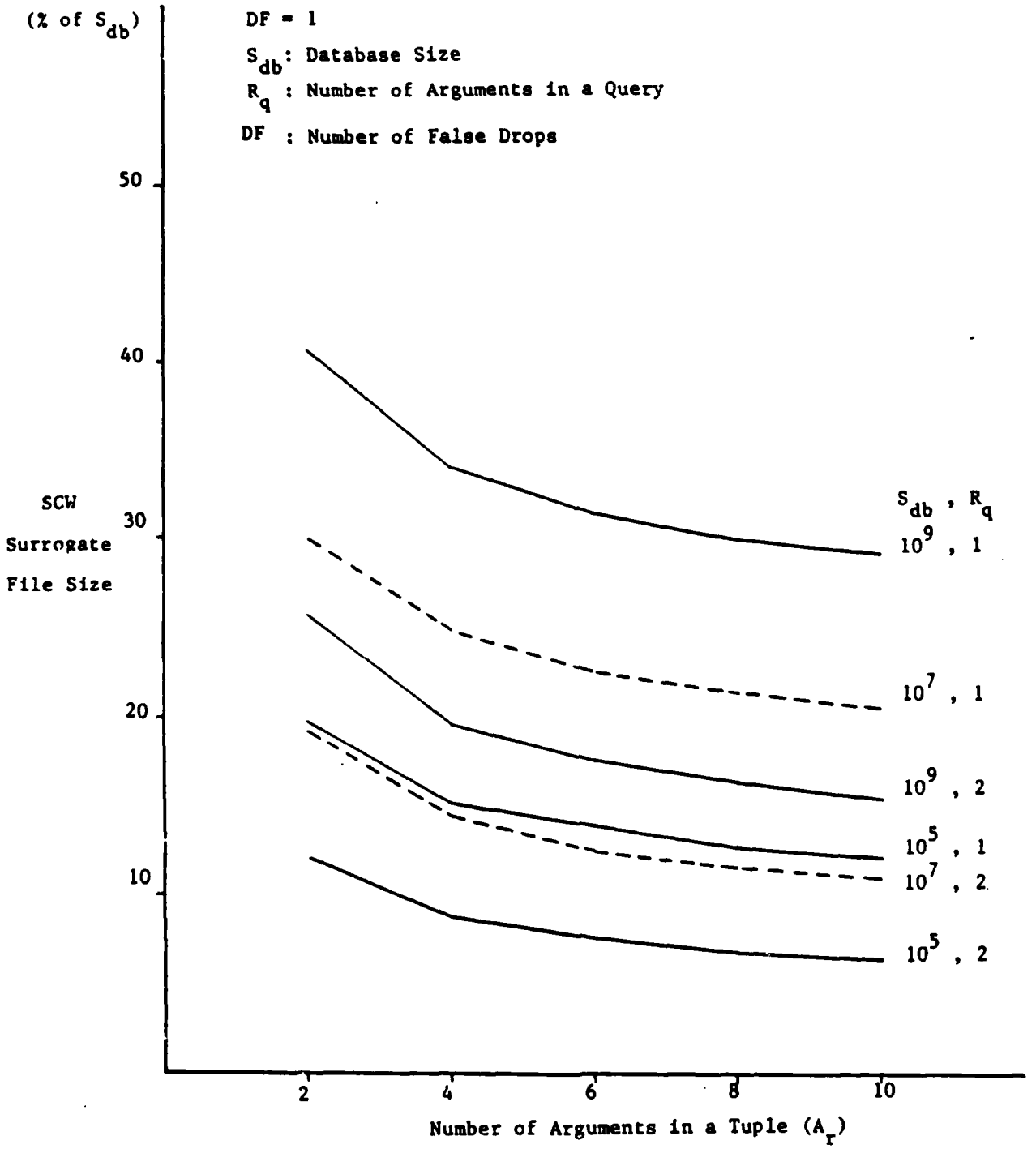


Figure 5.1 Effect of DB Size and the Average Number of Arguments in a Query on the SCW Surrogate File Size.

tions 4.1 to 4.3.

In Figure 5.2 we examine the effect of the number of false drops (DF). That is, if we allow more false drops then the length of the SCW becomes shorter which results in a smaller S_{scw}. However, more false drops leads to more database accesses.

In designing the SCW surrogate file one must set the expected number of arguments in a query. In terms of size, the worst case of course is when R_q is 1 and as the value for R_q is set at progressively higher values S_{scw} becomes very small as indicated in Figure 5.3. However, if we assume large R_q in designing the SCW file, we have to allow more false drops than the expected number of false drops, DF, whenever the number of arguments in a query is smaller than R_q. (Roberts 1979)

In Figure 5.4 we plot the size of the CCW surrogate file(S_{ccw}) as a function of the average redundancy(C_g) in the data. Note that with greater redundancy S_{ccw} becomes smaller because a smaller number of bits can be used for each binary code word. Also note that S_{db} and A_r have significant effects on S_{ccw}.

Finally, in Figure 5.5 we compare S_{scw} and S_{ccw} for various conditions. With regard to the size of surrogate files, we can say that the CCW file technique is better than the SCW technique, even though S_{scw} may be smaller than S_{ccw} when R_q is large, because we assumed that the average number of arguments in a query is usually not more than 2.

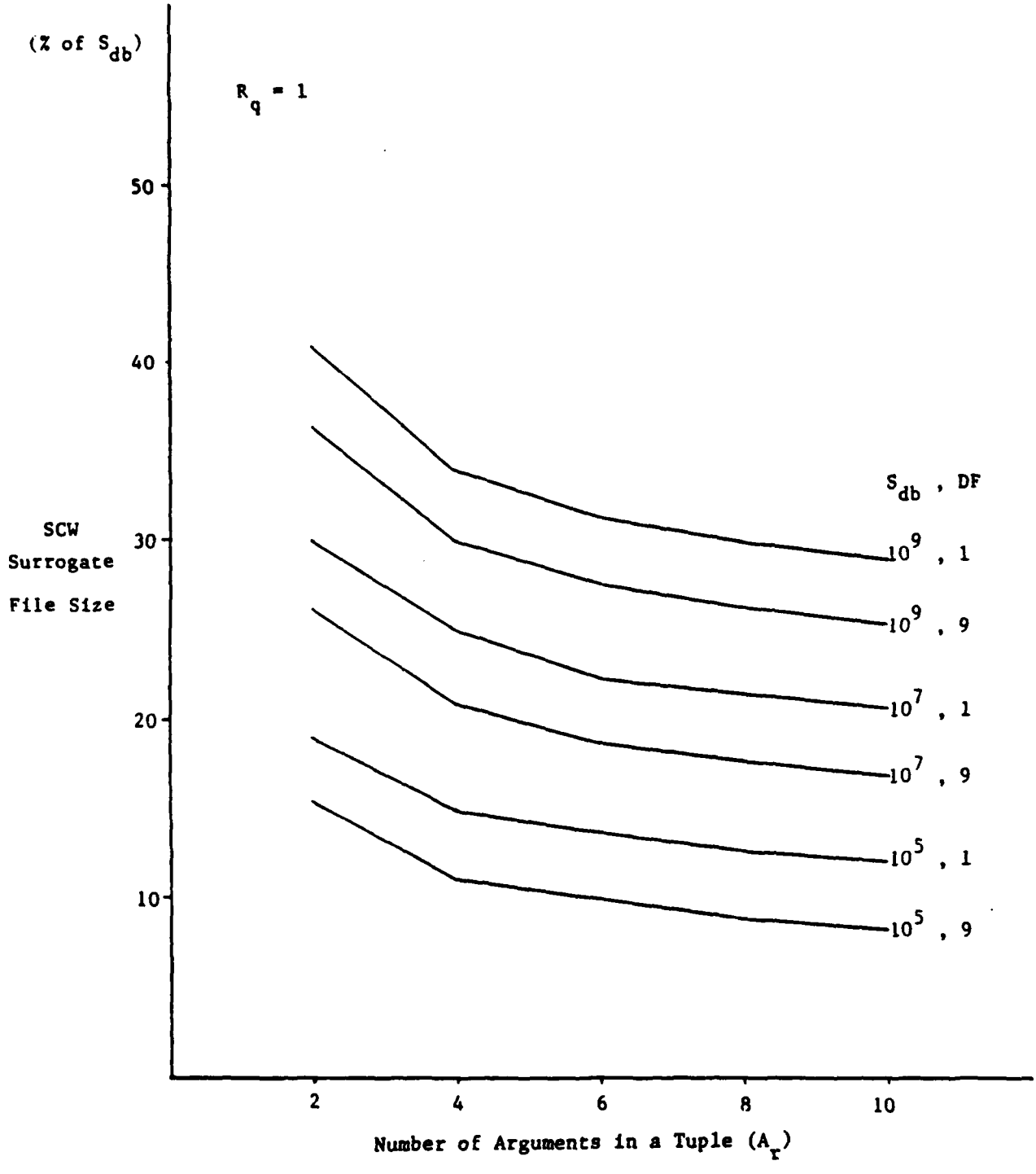


Figure 5.2 Effect of False Drops on the SCW Surrogate File Size

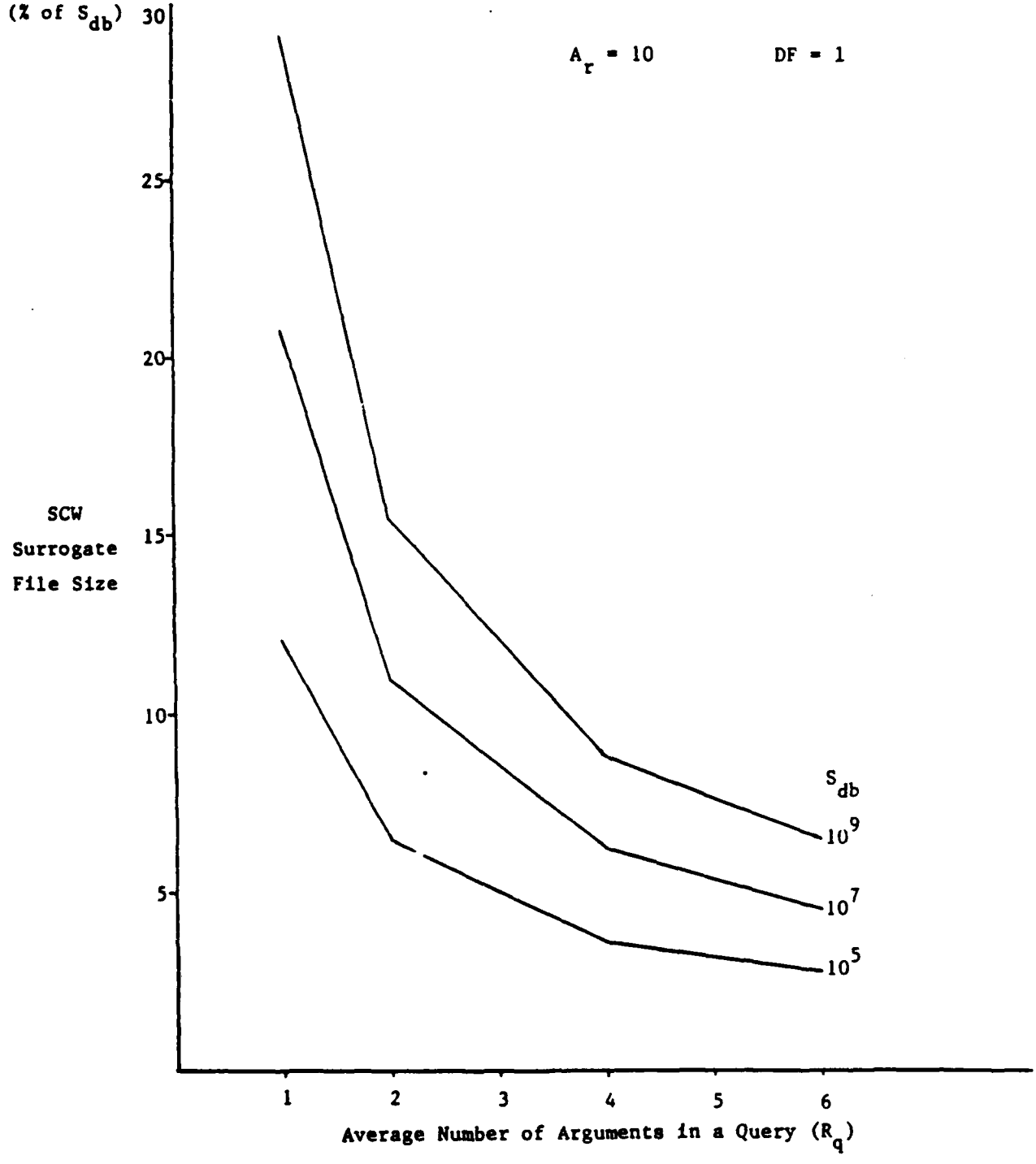


Figure 5.3 Effect of the Average Number of Arguments in a Query on the SCW Surrogate File Size.

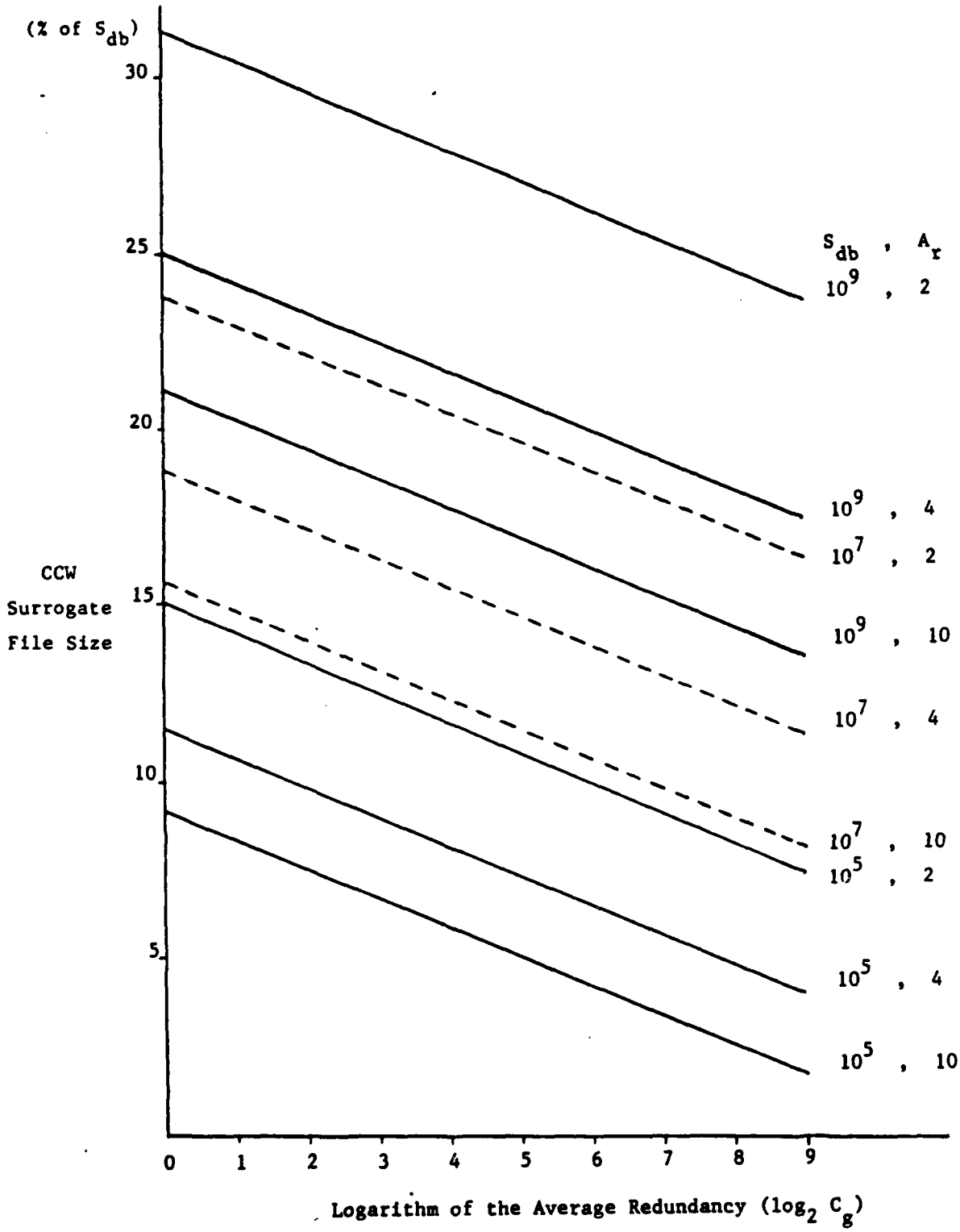


Figure 5.4 Effect of the Average Redundancy on the CCW Surrogate File Size.

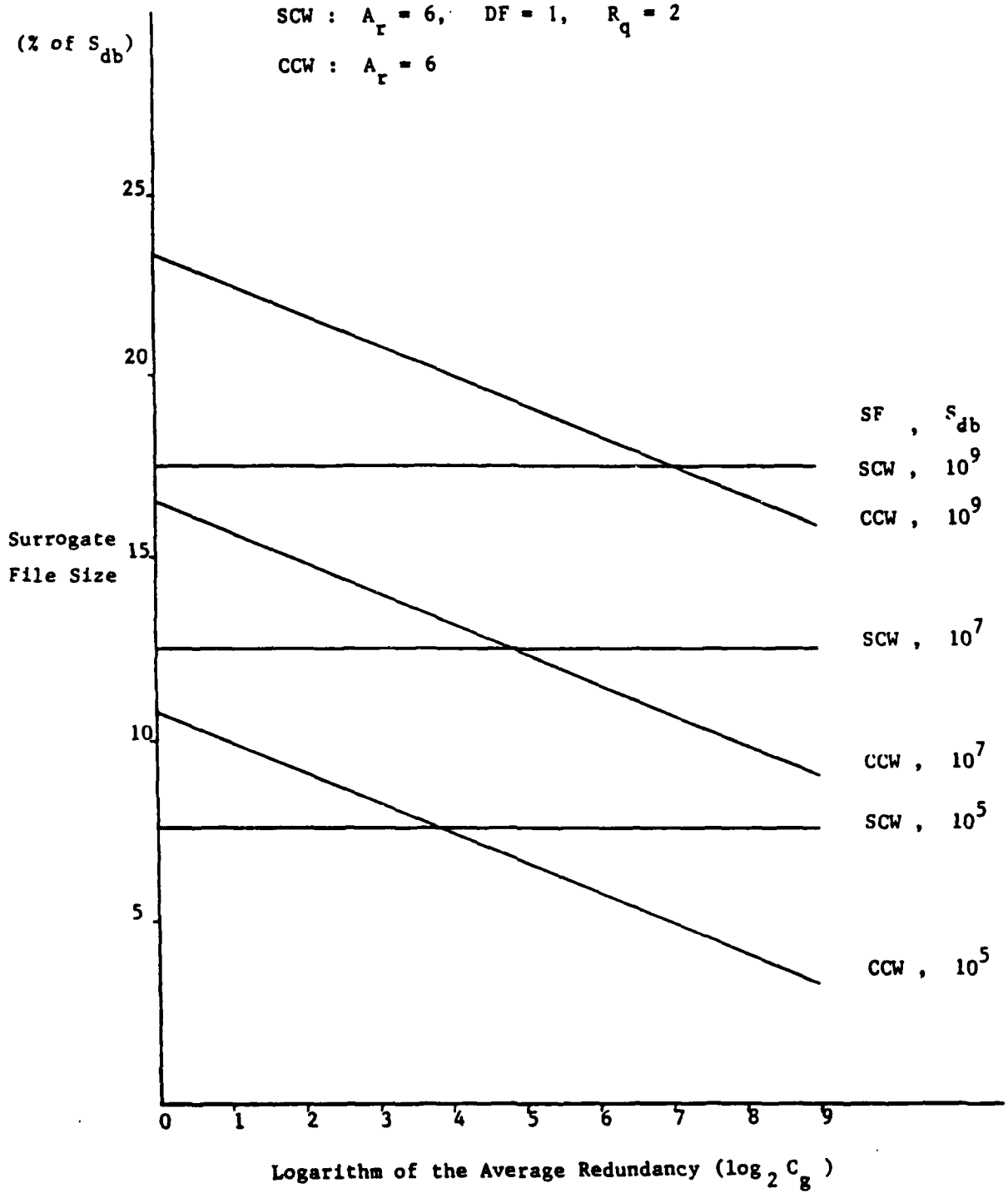


Figure 5.5 SCW and CCW Surrogate File Size Comparison

5.2. Query Response Time

For the query response time, it is assumed that a partial-match query is performed and the BCW of the surrogate file is compared with the QCW by using byte by byte comparison. The results are shown in Figures 5.6 through 5.29 and Table 5.1 shows the values of parameters used in this simulation. The parameters relating to the disk are obtained from the characteristics of the DEC RA81 disk. (Kridle 1983)

Query response time of CCW and SCW techniques are simulated based on the equations developed in the previous section and we obtained the following results.

In Figures 5.6 through 5.8 and 5.9 through 5.11, we plot the query response times, QT_{scw} and QT_{ccw}, and corresponding subprocessing times for S_{db} of 10^5, 10^7 , and 10^9 bytes, respectively. When S_{db} is 10^5 bytes, most of the query response time is spent for database access. But when S_{db} is 10^9 bytes, the query response time becomes very large and most of the query response time is spent for surrogate file accessing and searching because of the increased surrogate file size and sequential searching of the surrogate file.

In Figures 5.12 through 5.14, we plot QT_{scw} as a function of A_r and DF, and in Figures 5.15 through 5.17, we plot QT_{ccw} as a function of A_r and C_g. We can see from those figures that the A_r affects QT_{scw} and QT_{ccw} little because we assumed that the S_{db} remains constant under the variation of A_r.

In Figures 5.18 through 5.20, we plot the QT_{scw} as a function of R_q and DF. We can see from those figures that when S_{db} is 10^5 bytes, R_q is not a factor which affects QT_{scw}, but QT_{scw} increases as DF increases. However, when S_{db} is 10^9 bytes, the result is reversed, that is, R_q affects the QT_{scw} considerably while DF

Parameter	Value
Average seek time	28 msec
Minimum seek time	6 msec
Rotational speed	3600 RPM
Data transfer rate	2000 bytes/msec
Time for byte comparison	3 μsec

Table 5.1 The Values of Parameters Used in the Simulation

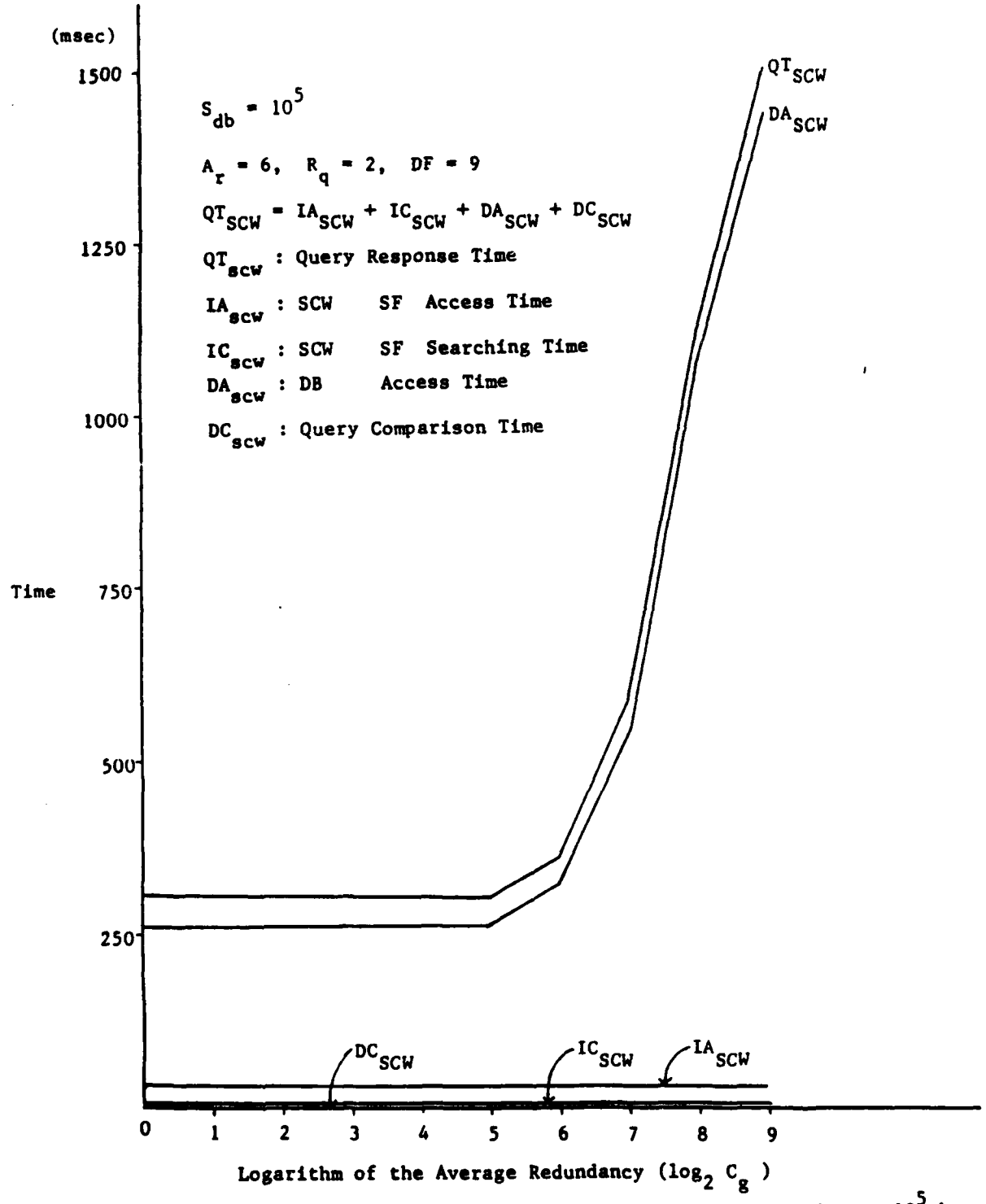


Figure 5.6 Components of the SCW Query Response Time ($S_{db} = 10^5$ bytes)

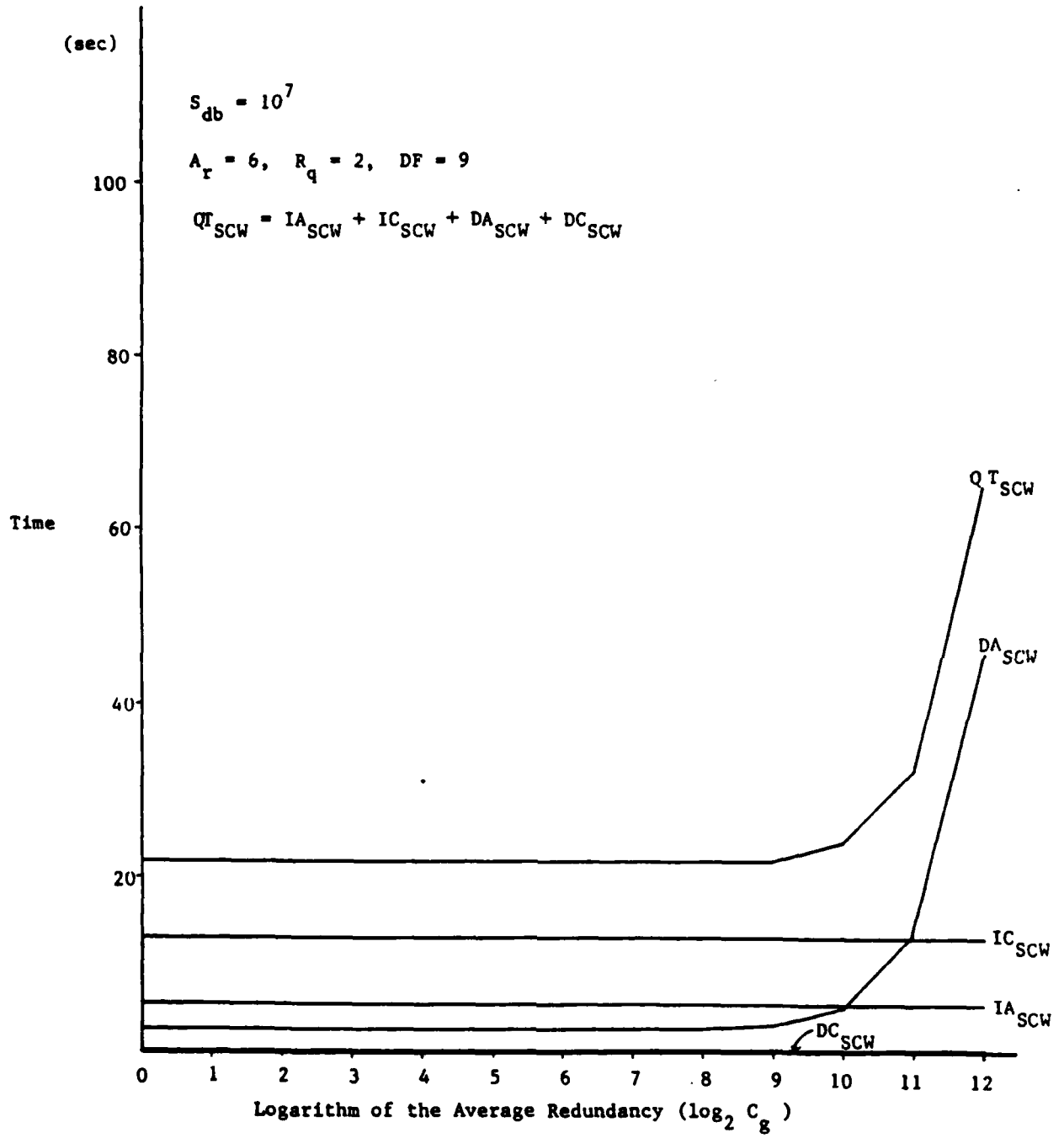


Figure 5.7 Components of the SCW Query Response Time ($S_{db} = 10^7$ bytes)

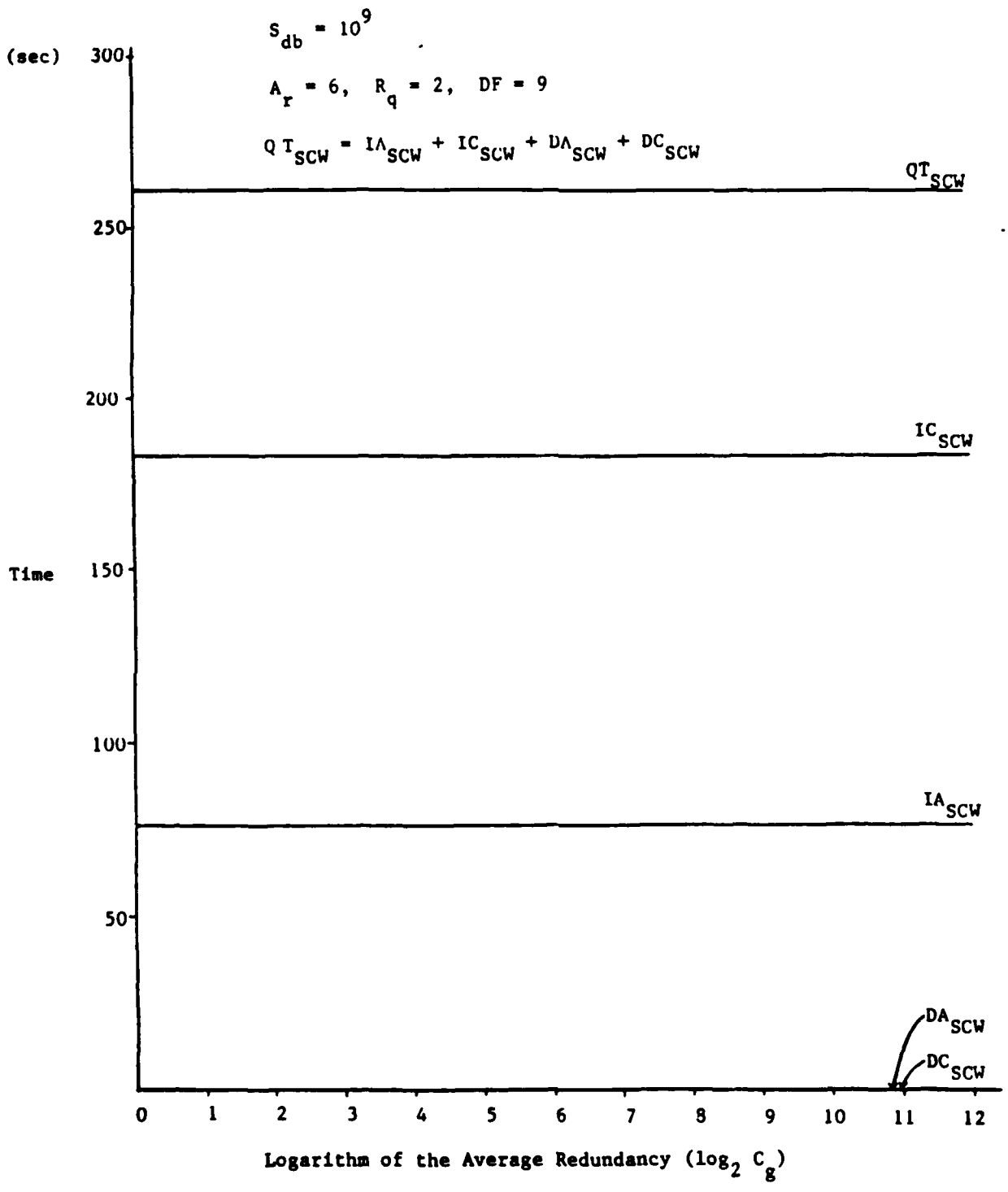


Figure 5.8 Components of the SCW Query Response Time ($S_{db} = 10^9$ bytes)

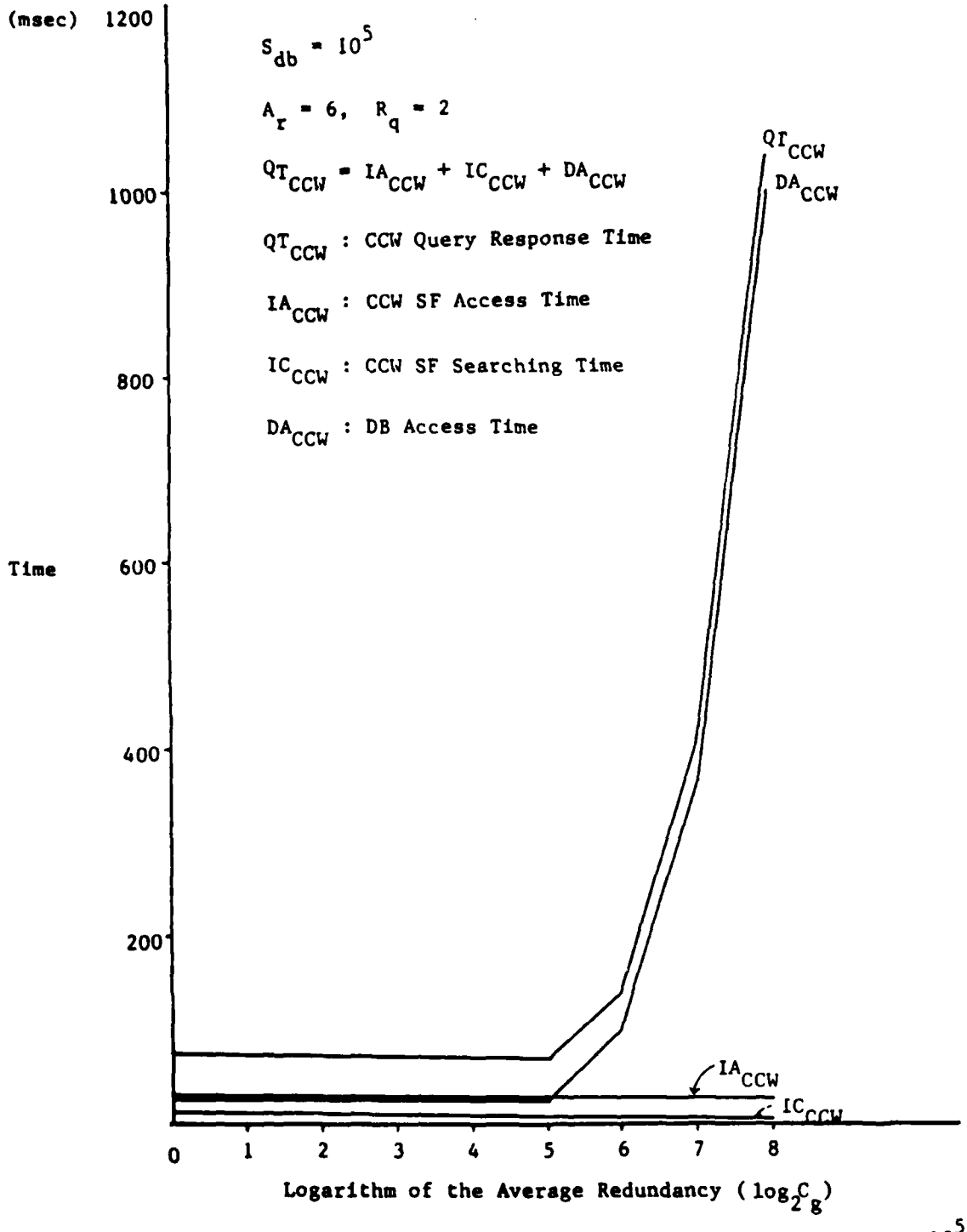


Figure 5.9 Components of the CCW Query Response Time ($S_{db} = 10^5$ bytes)

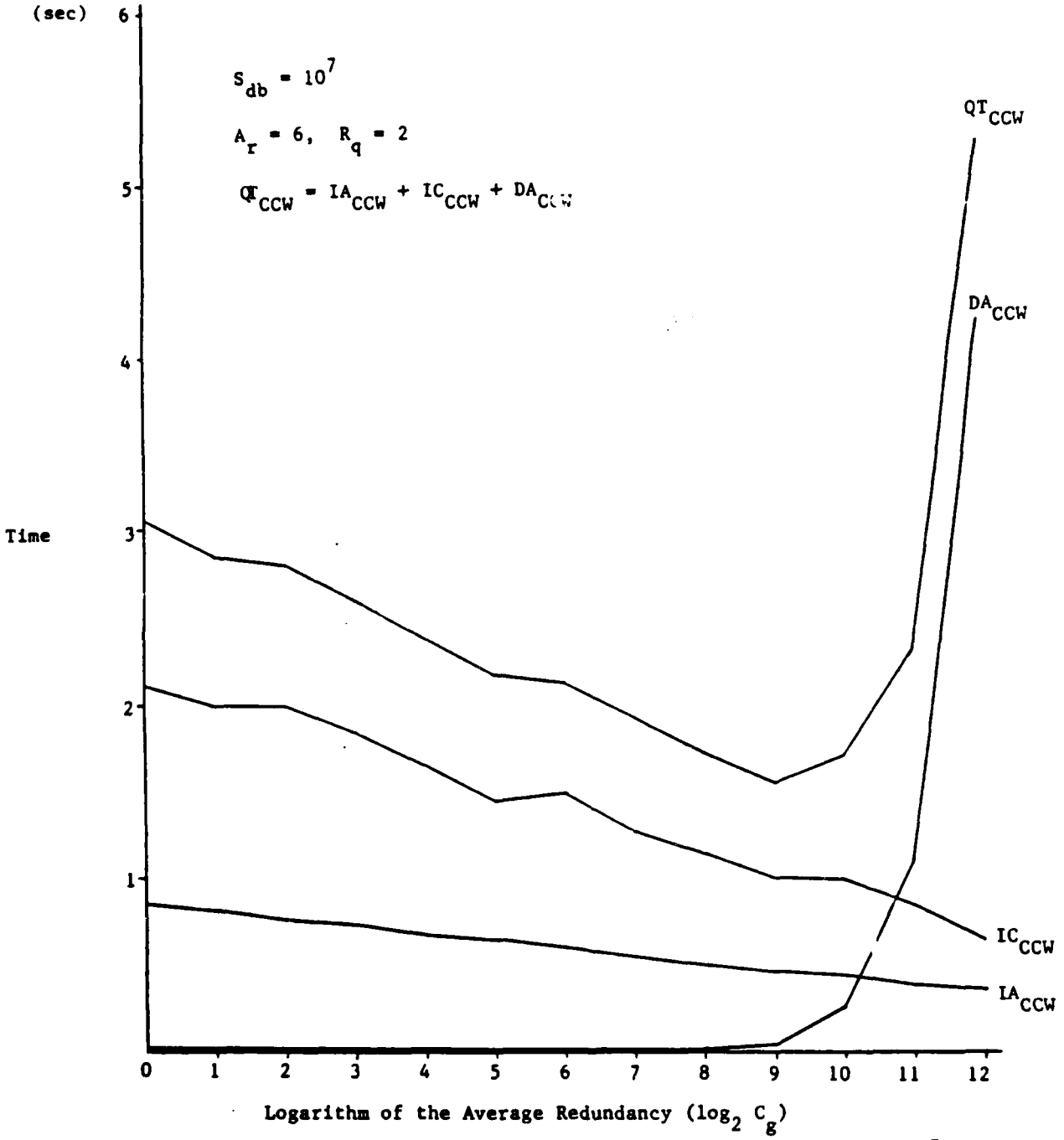


Figure 5.10 Components of the CCW Query Response Time ($S_{db} = 10^7$ bytes)

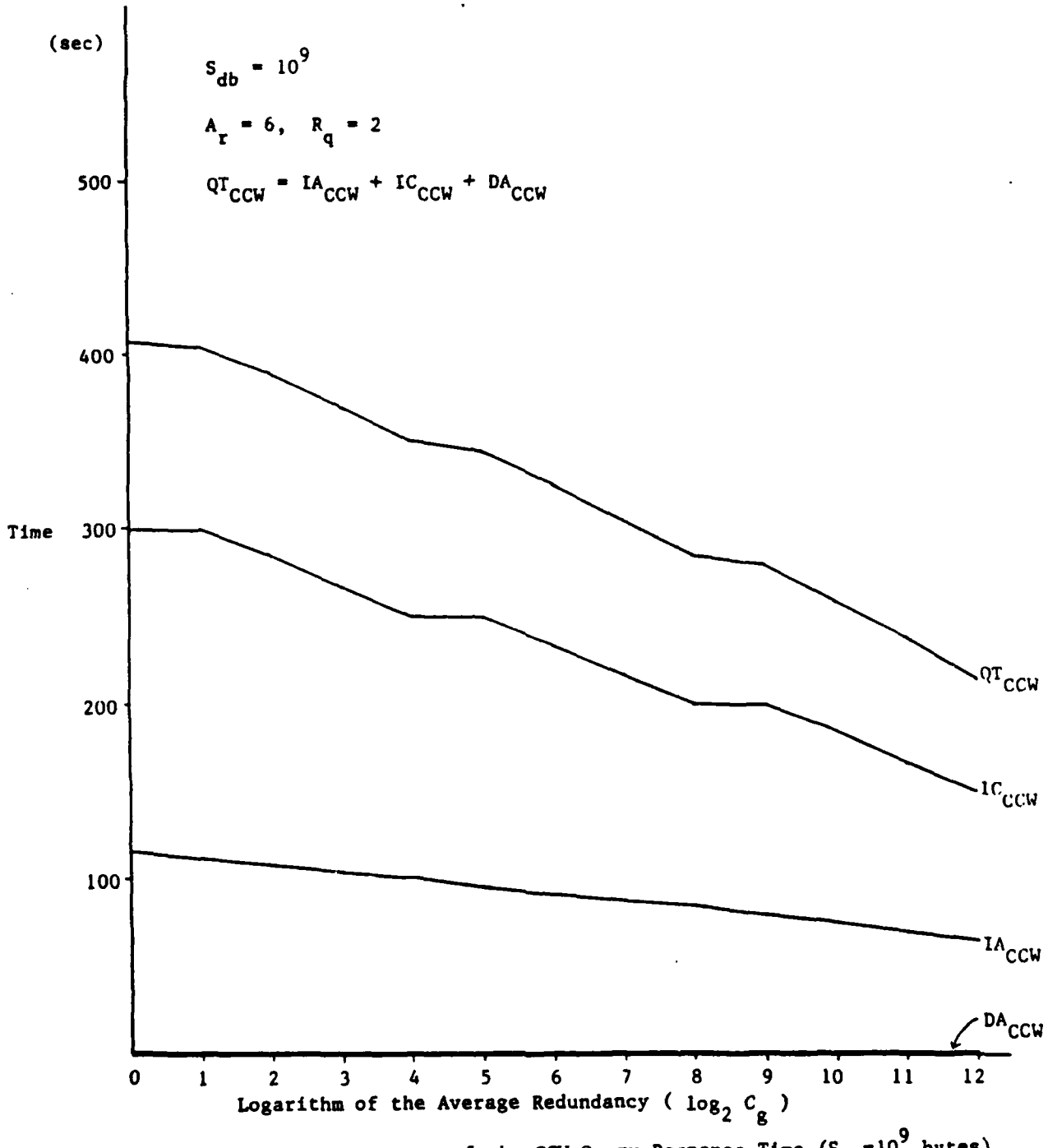


Figure 5.11 Components of the CCW Query Response Time ($S_{db}=10^9$ bytes)

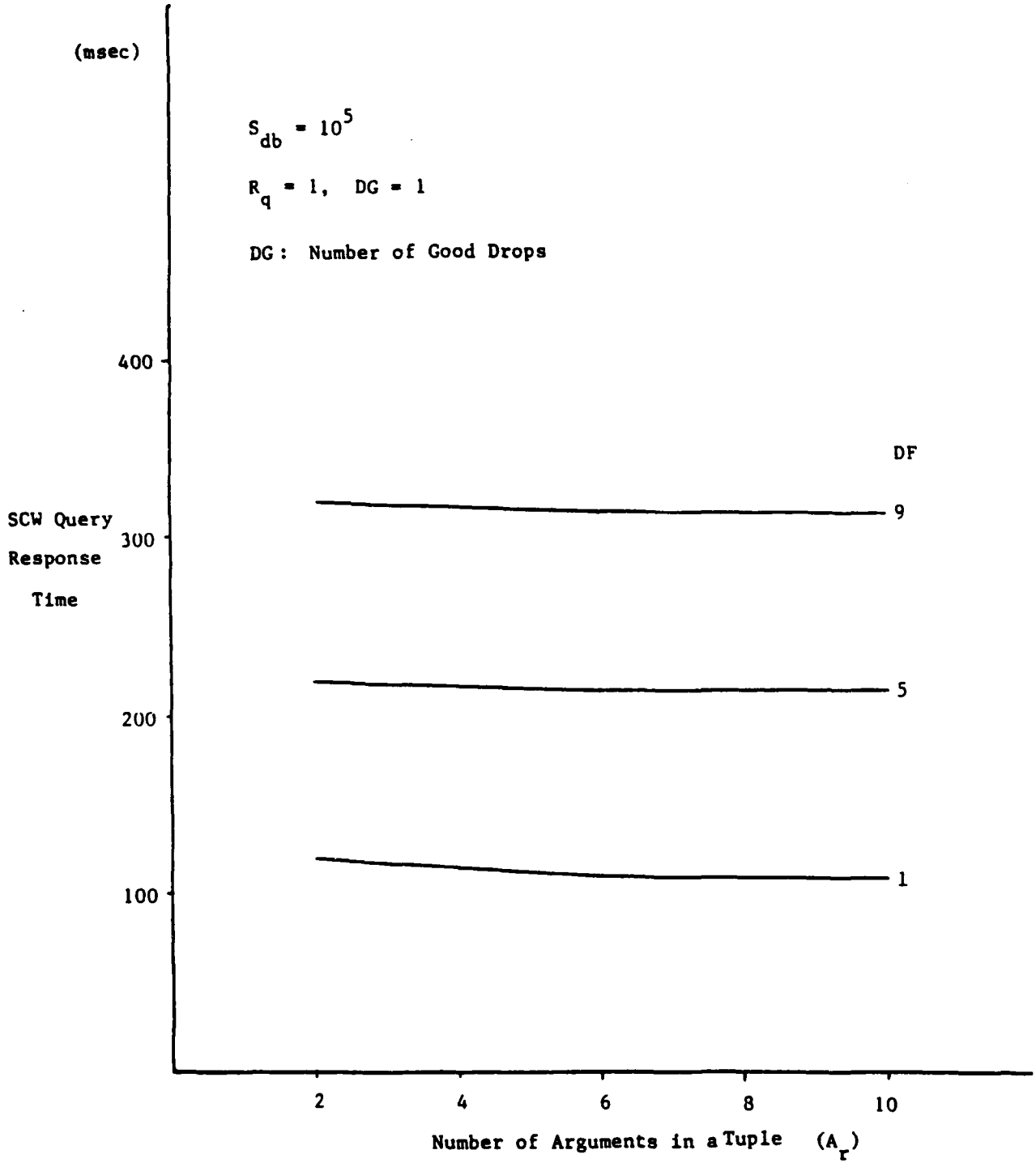


Figure 5.12 Effect of the Number of Arguemnts in a Tuple on the SCW Query Response Time ($S_{db} = 10^5$ bytes)

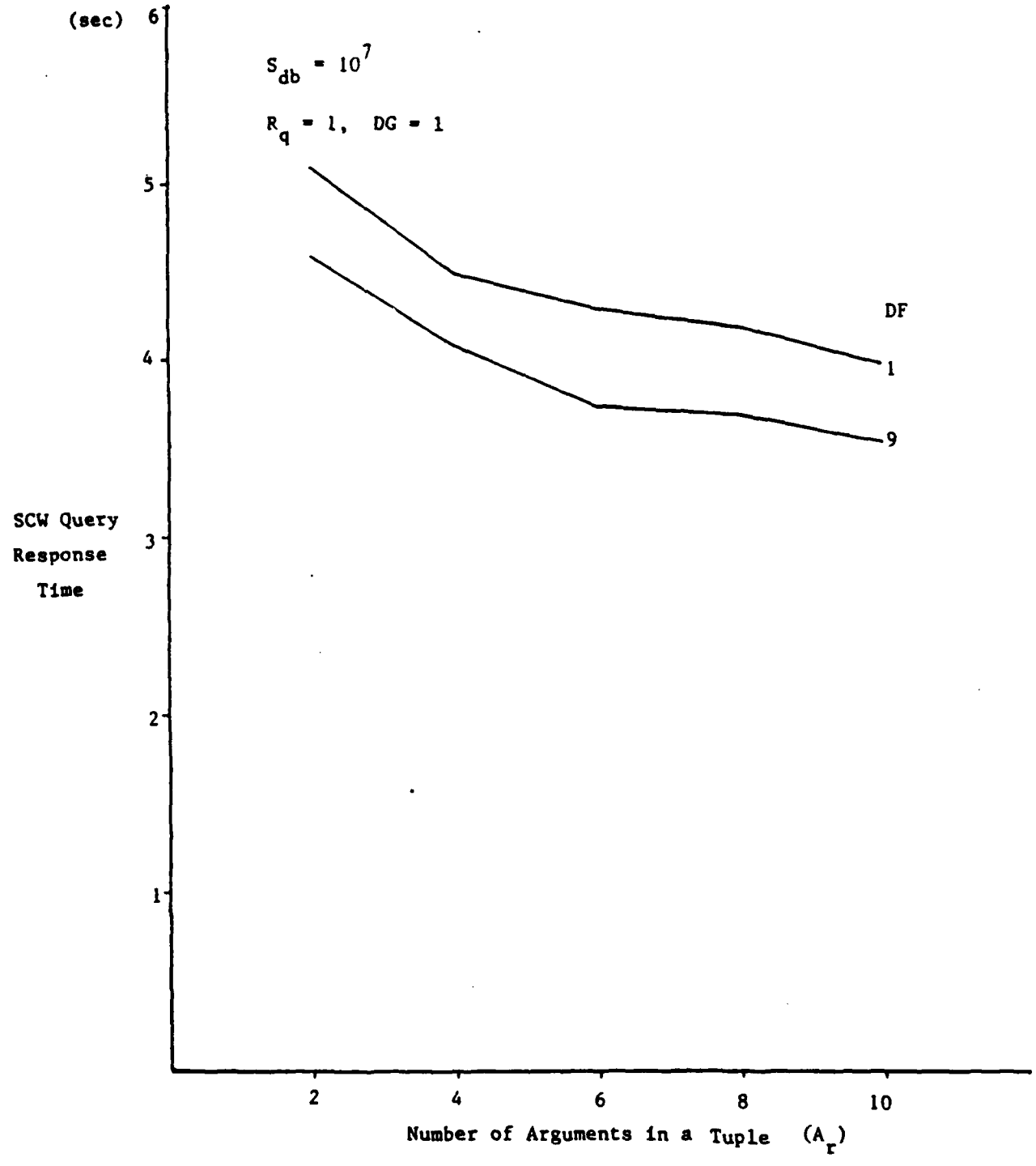


Figure 5.13 Effect of the Number of Arguments in a Tuple on the SCW Query Response Time ($S_{db} = 10^7$ bytes)

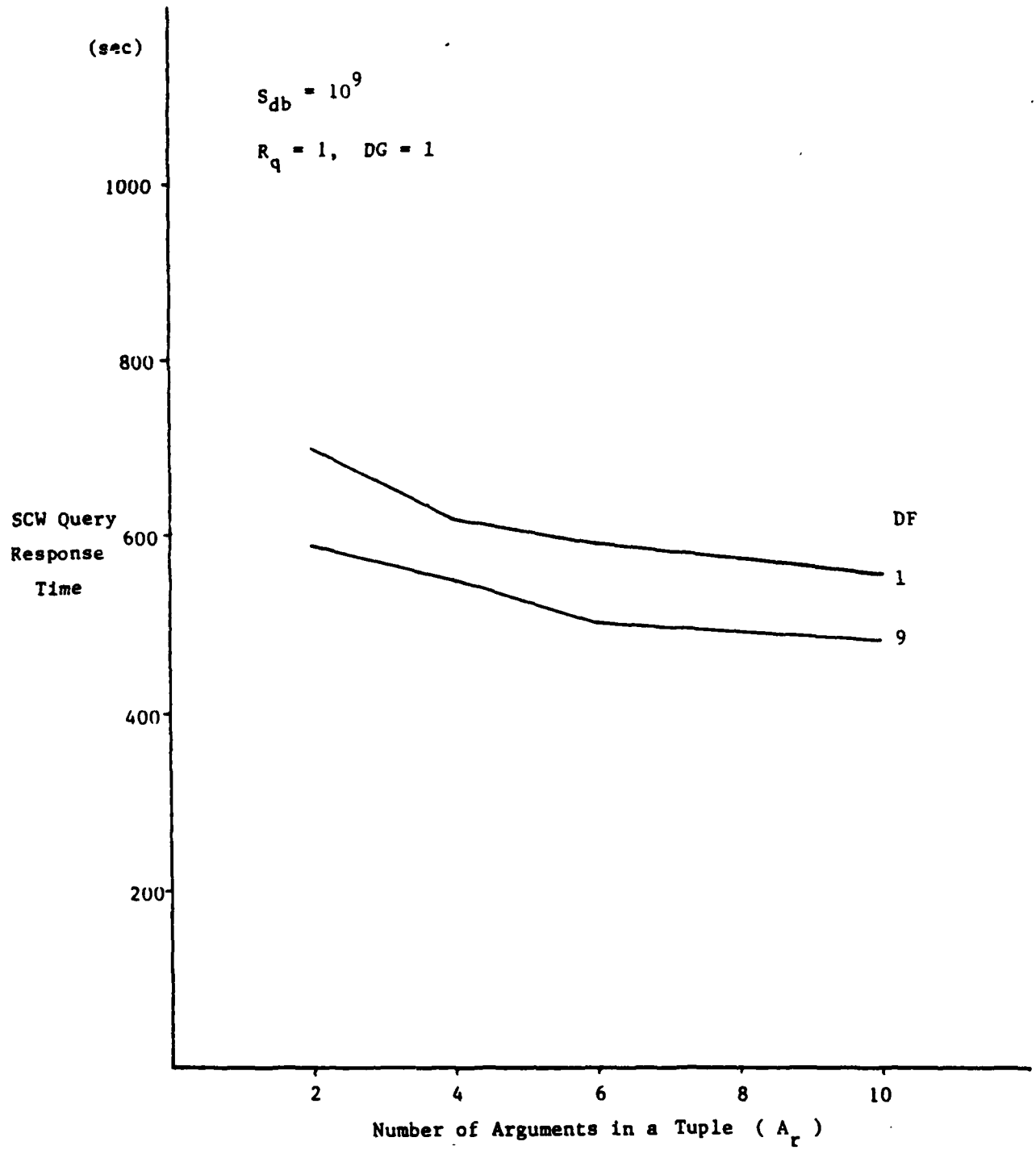


Figure 5.14 Effect of the Number of Arugments in a Tuple on the SCW Query Response Time ($S_{db} = 10^9$ bytes)

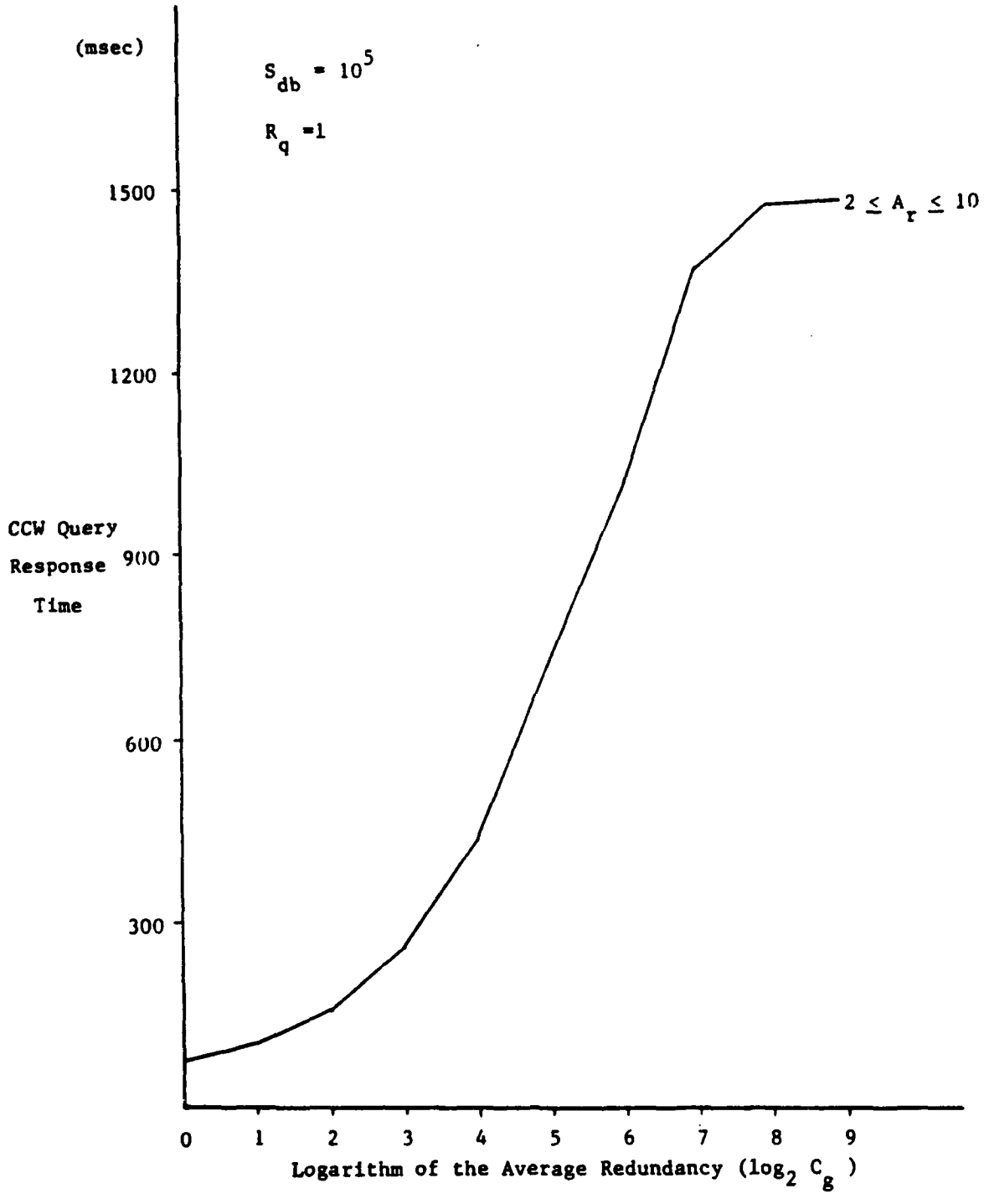


Figure 5.15 Effect of the Number of Arguments in a Tuple on the CCW Query Response Time ($S_{db} = 10^5$ bytes)

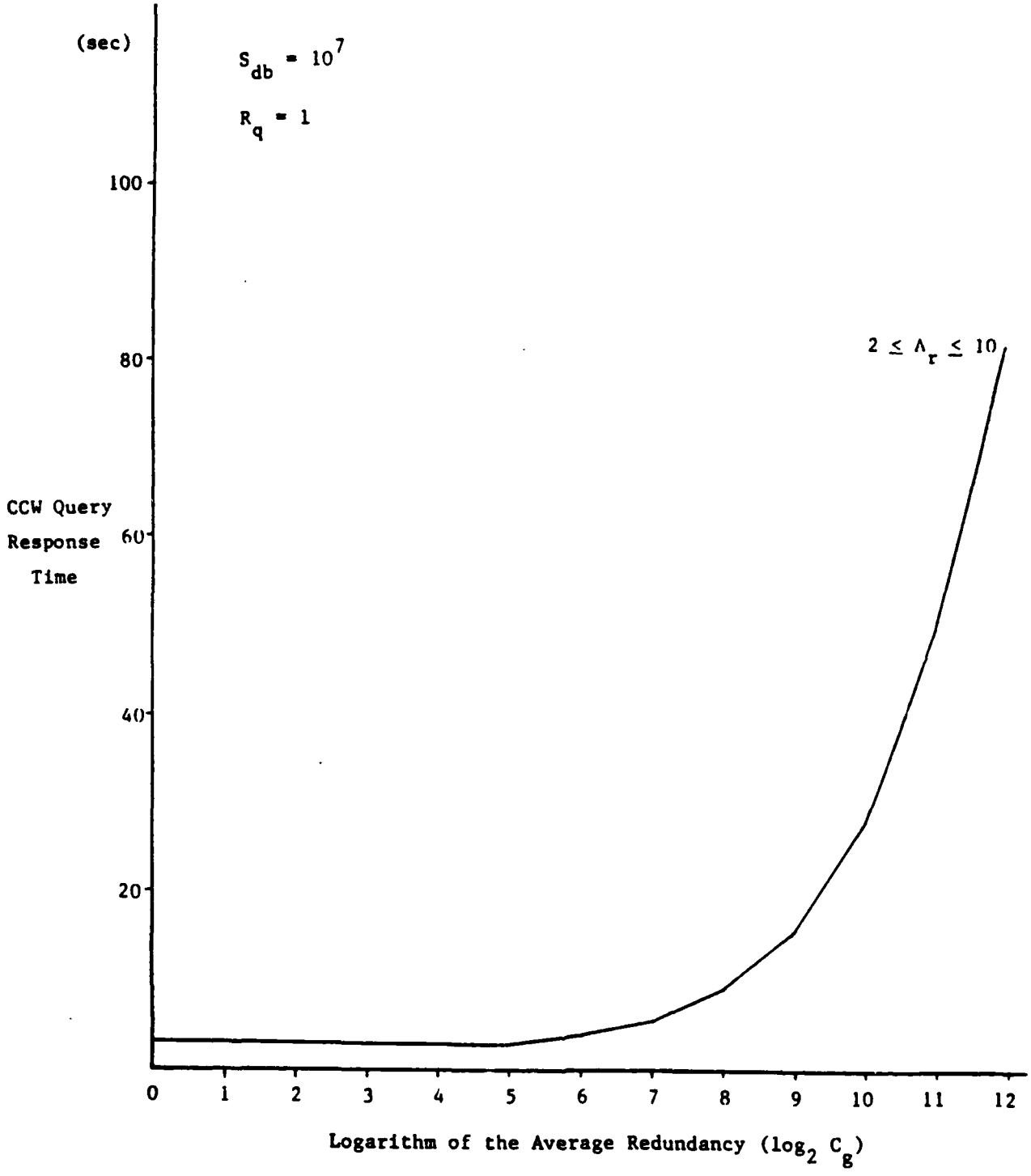


Figure 5.16 Effect of the Number of Arguments in a Tuple on the CCW Query Response Time ($S_{db} = 10^7$ bytes)

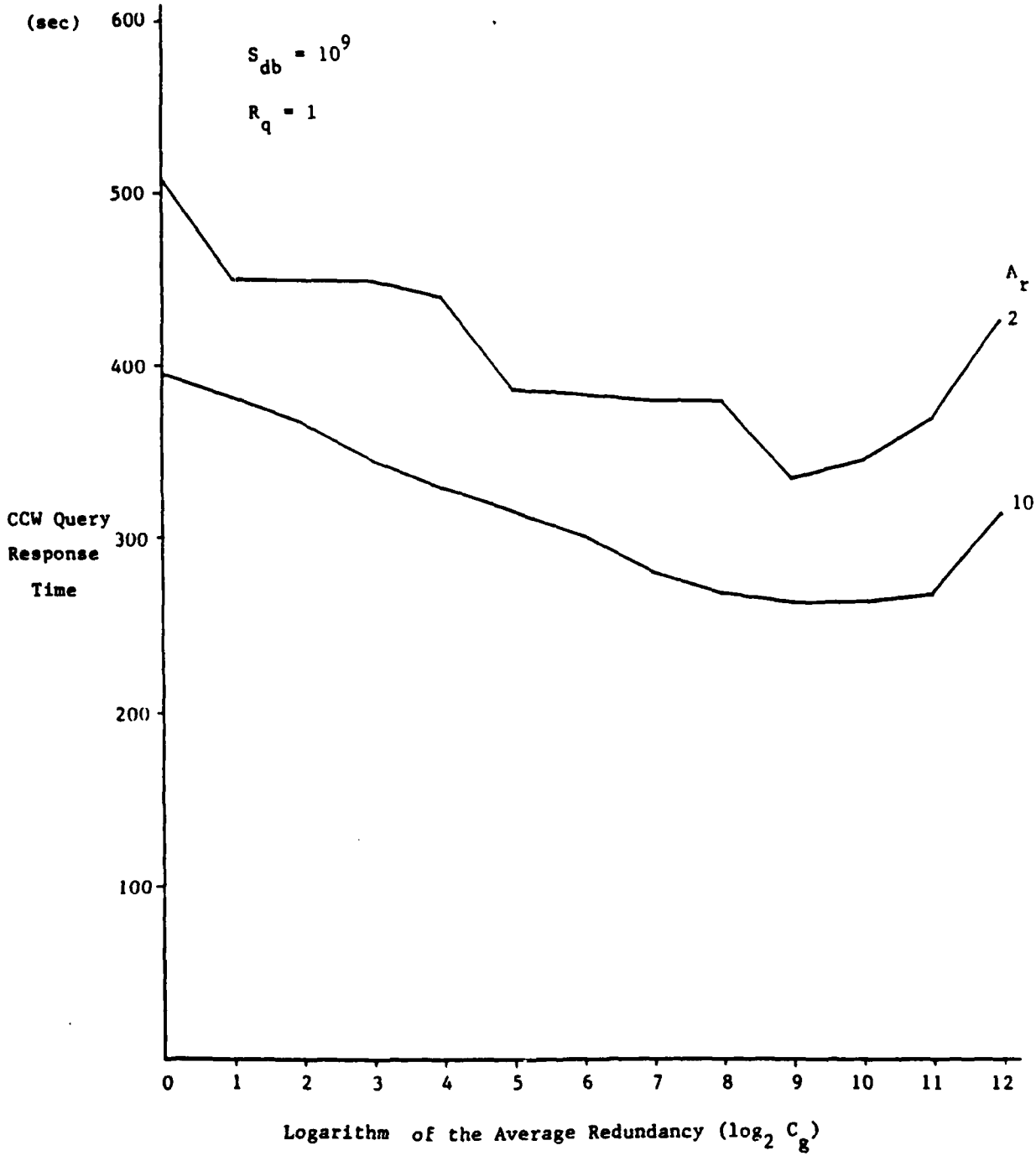


Figure 5.17 Effect of the Number of Arguments in a Tuple on the CCW Qeury Response Time ($S_{db} = 10^9$ bytes)

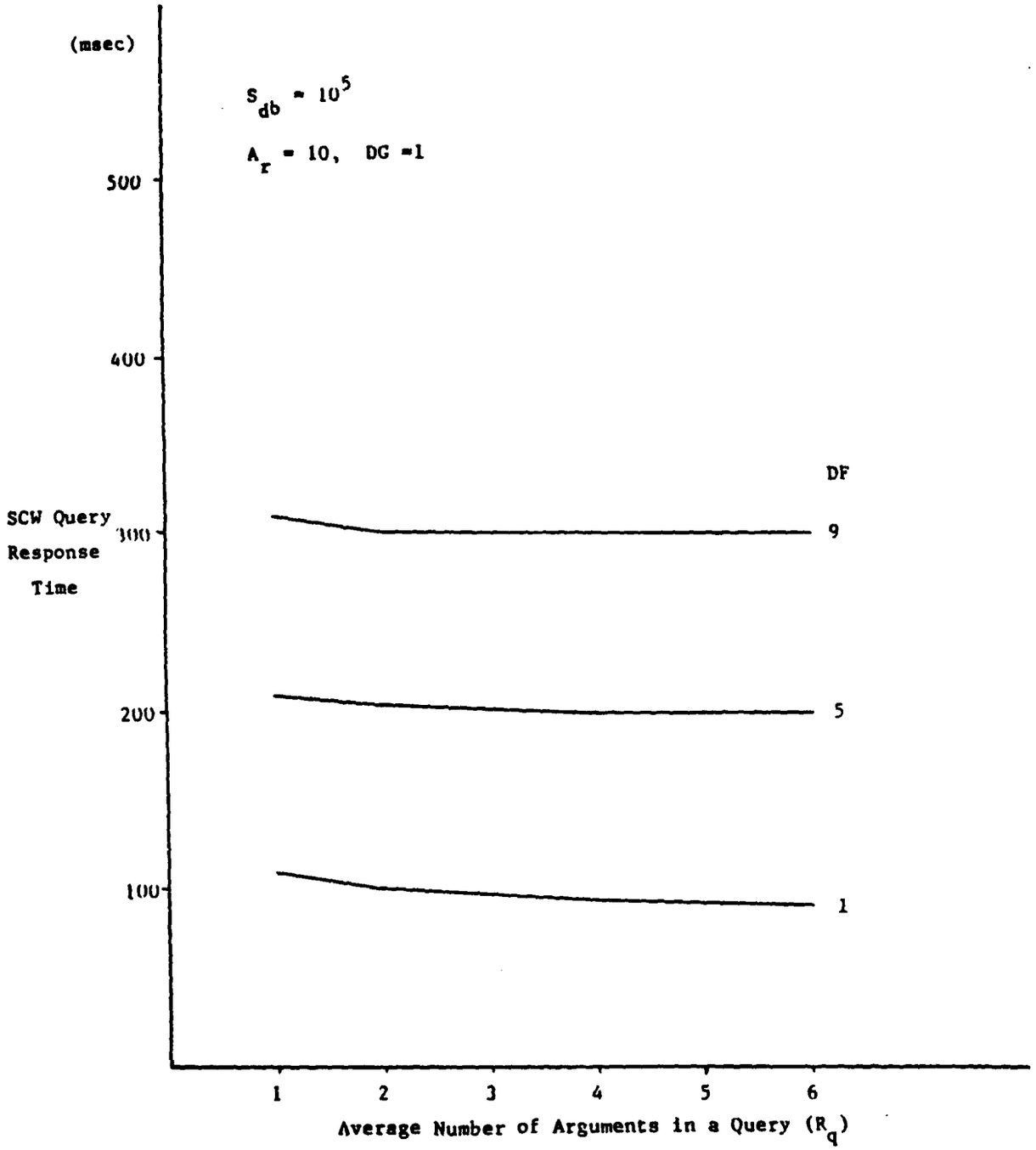


Figure 5. 18 Effect of False Drops and the Average Number of Arguments in in a Query on the SCW Query Response Time ($S_{db} = 10^5$ bytes)

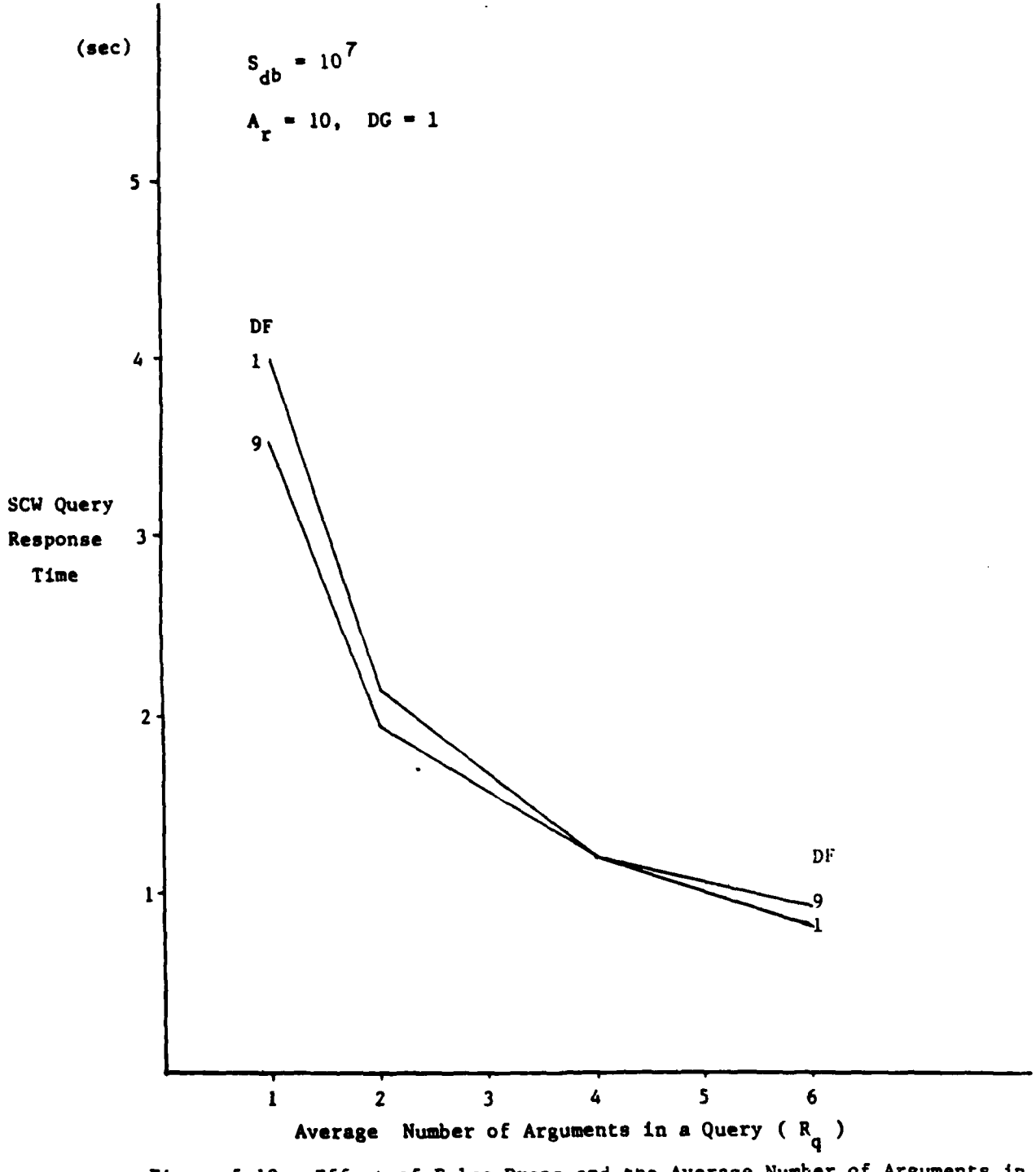


Figure 5.19 Effect of False Drops and the Average Number of Arguments in a Query on the SCW Query Response Time ($S_{db} = 10^7$ bytes)

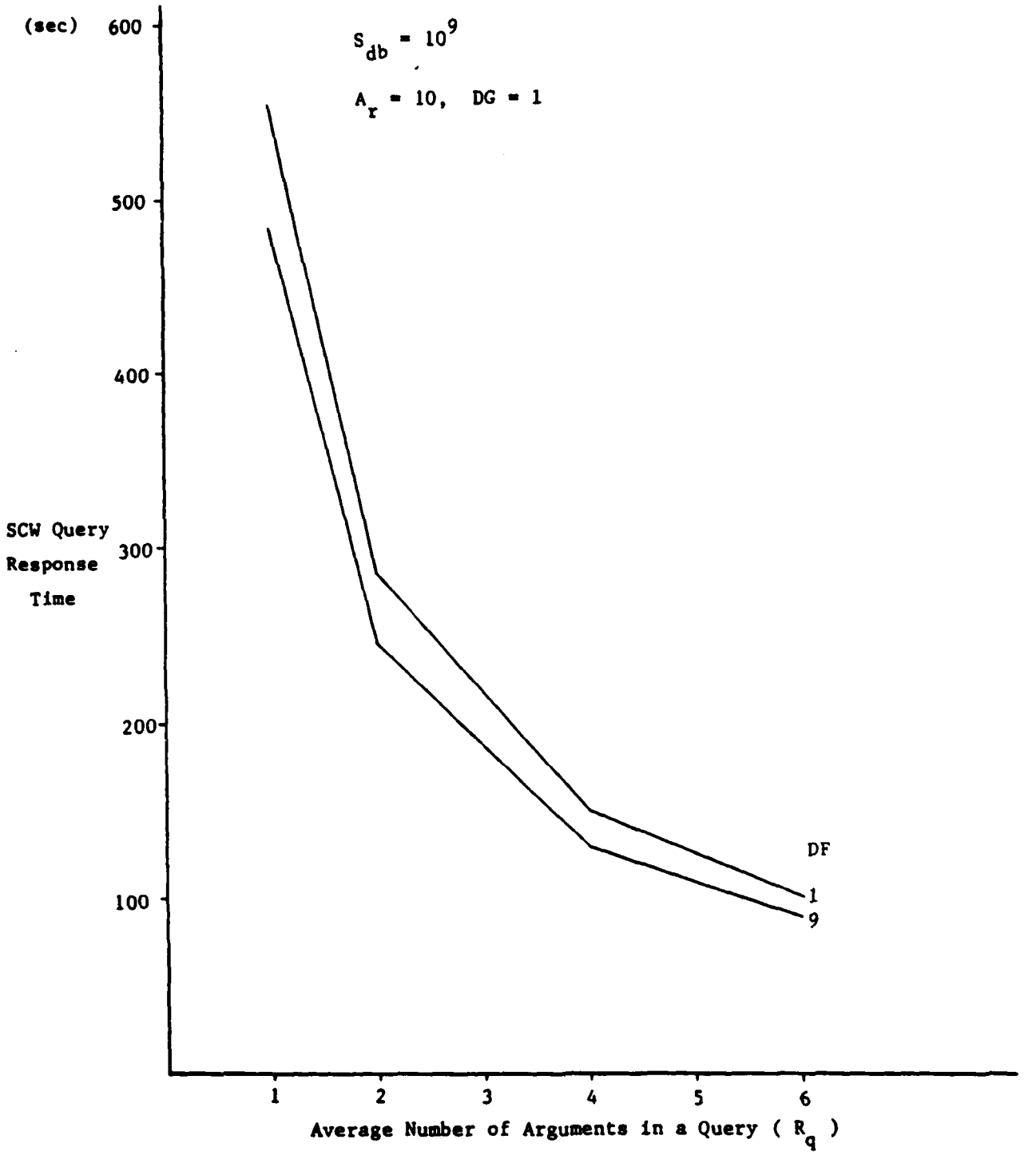


Figure 5.20 Effect of False Drops and the Average Number of Arguments in a Query on the SCW Query Response Time ($S_{db} = 10^9$ bytes)

does not. There are two reasons supporting this result:

1) S_{scw} decreases as R_q increases. However, when S_{db} is small, S_{scw} is also small for any R_q so that the time for accessing and searching the SCW file is almost constant. Therefore, the time for accessing the database, which depends on DF, becomes a major factor in QT_{scw}.

2) When S_{db} is large, S_{scw} becomes large so that most of QT_{scw} is spent for accessing and searching the SCW file. Therefore, S_{scw} is a main factor deciding QT_{scw}. Since S_{scw} largely depends on R_q, the change in R_q is directly reflected in QT_{scw}.

We plot the QT_{scw} and QT_{ccw} as a function of R_q and C_g in Figures 5.21 through 5.23 and 5.24 through 5.26, respectively. From Figures 5.21 and 5.24 we can see that QT_{scw} and QT_{ccw} are largely affected by C_g when S_{db} is 10^5 bytes and R_q is small. However, as R_q becomes large, the effect of C_g on QT_{ccw} and QT_{scw} decreases. This fact is well explained by the role of R_q and C_g in determining the number of good drops:

1) If R_q is small and C_g is large, then there are so many good drops that a large amount of time is required for accessing the database.

2) If R_q becomes large, the number of good drops decrease considerably, and so does the database access time, which is the major component of the query response time when S_{db} is 10^5 bytes.

From Figures 5.23 and 5.26, we can see that when S_{db} is 10^9 bytes, as C_g increases QT_{scw} remains constant while QT_{ccw} decreases. This is because as S_{db} becomes large, the surrogate file size grows so fast that most of the query response time is taken for accessing and searching the surrogate file, i.e. the surrogate file size determines the query response time, however, S_{scw} does not

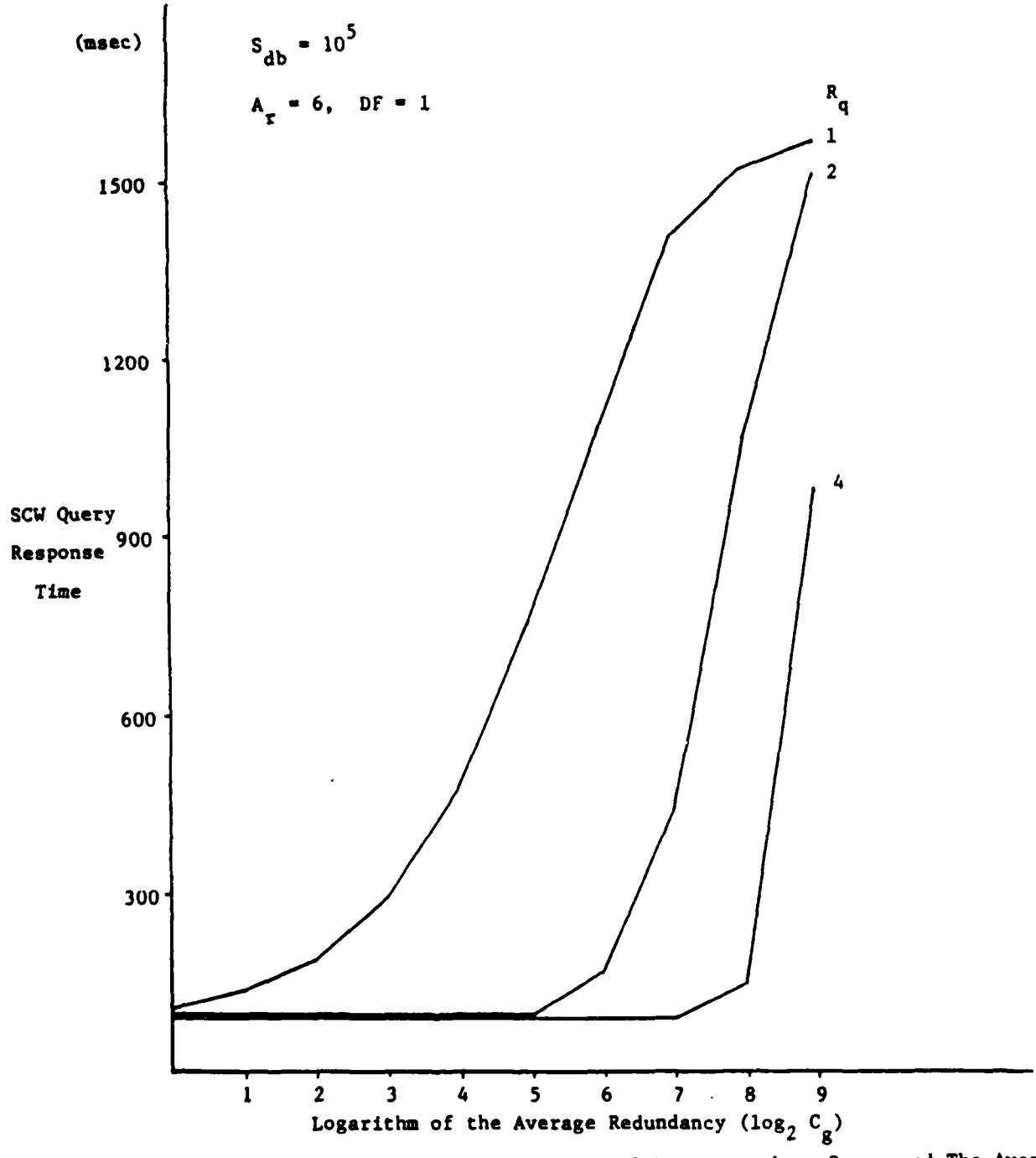


Figure 5.21 Effect of the Average Number of Arguments in a Query and The Average Redundancy on the SCW Query Response Time ($S_{db} = 10^5$ bytes)

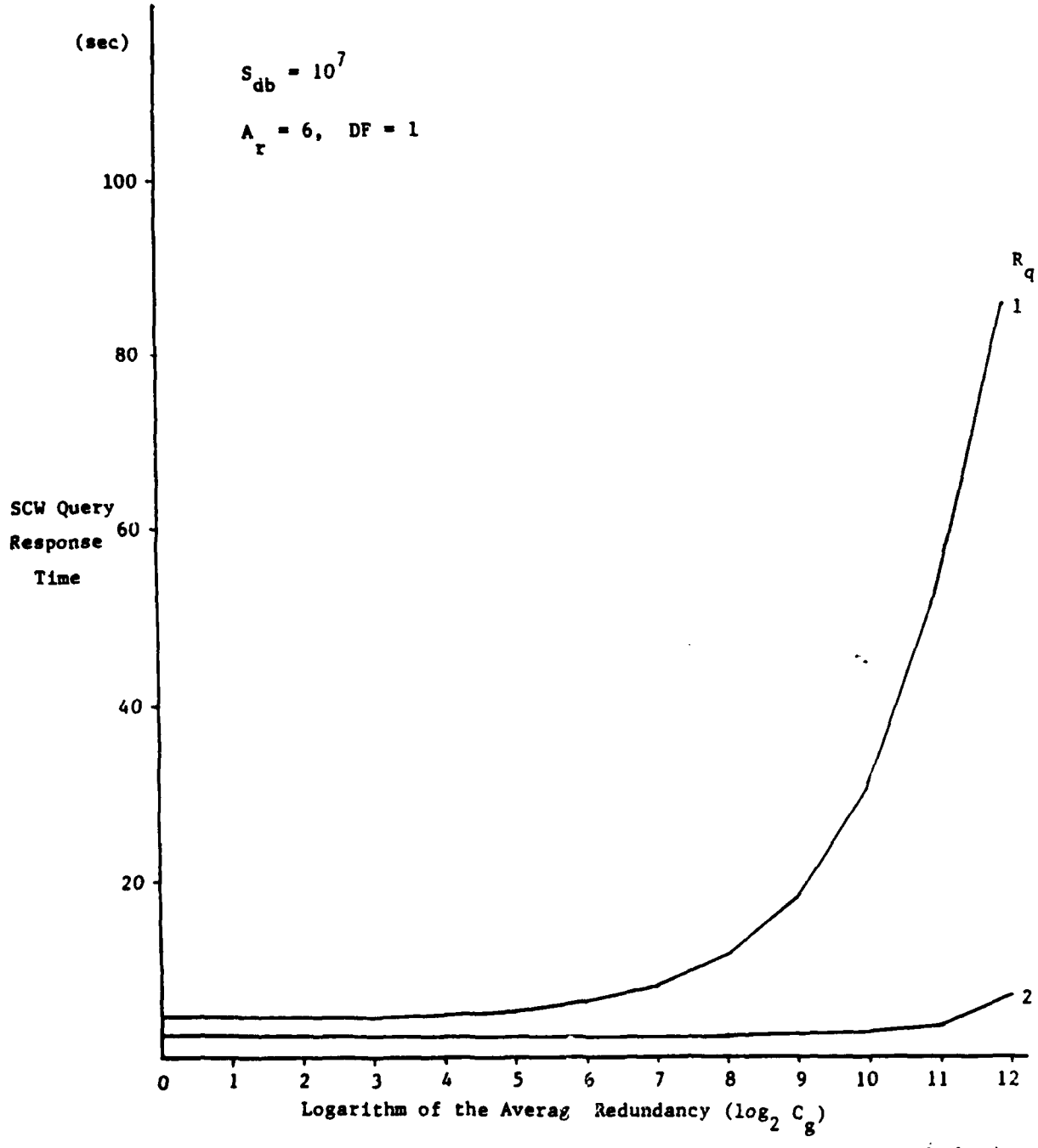


Figure 5.22 Effect of the Average Number of Arguments on a Query and the Average Redundancy on the SCW Query Response Time ($S_{db} = 10^7$ bytes)

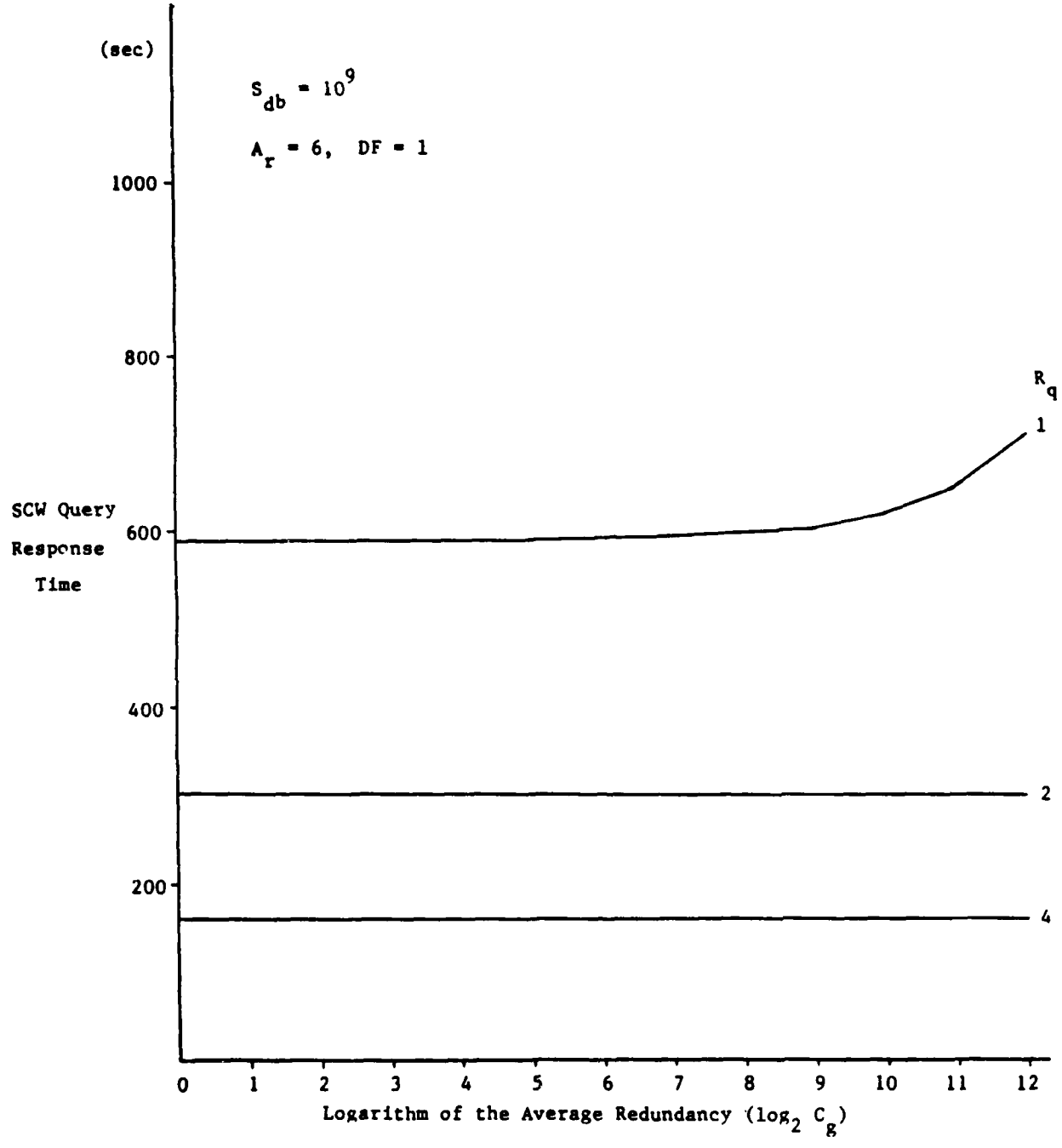


Figure 5.23 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the SCW Query Response Time ($S_{db} = 10^9$ bytes)

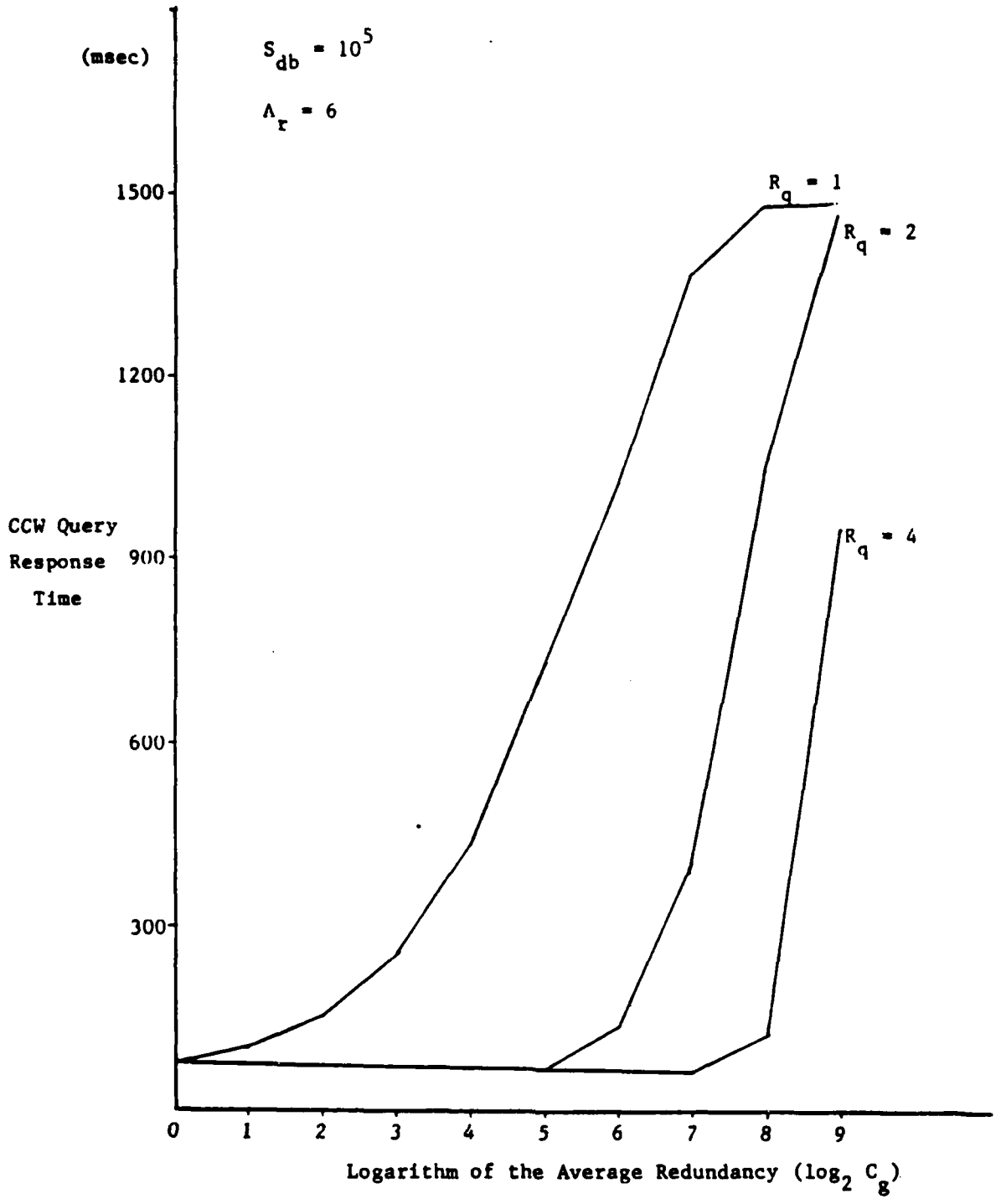


Figure 5.24 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the CCW Query Response Time ($S_{db} = 10^5$ bytes)

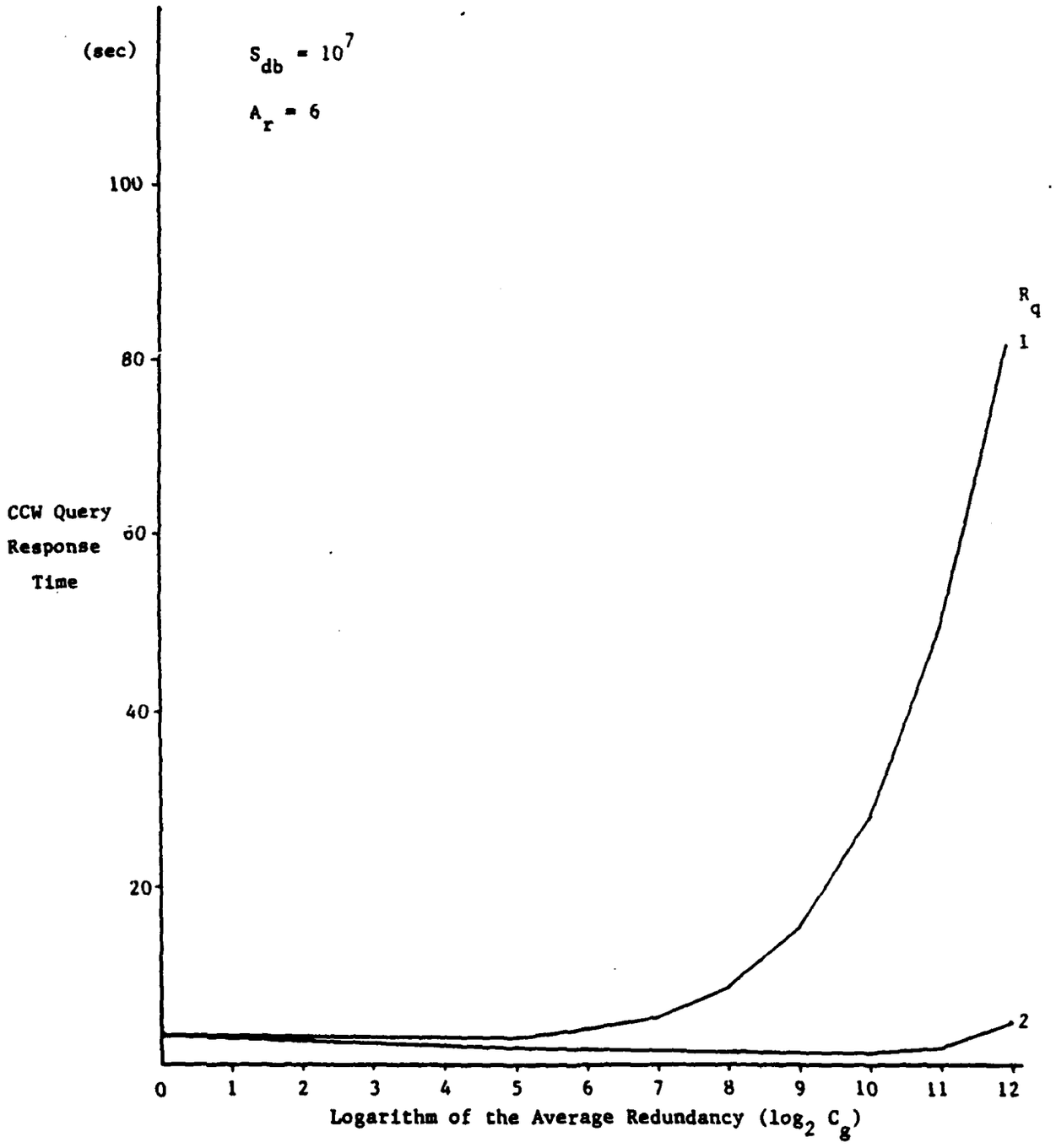


Figure 5.25 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the CCW Query Response Time ($S_{db} = 10^7$ bytes)

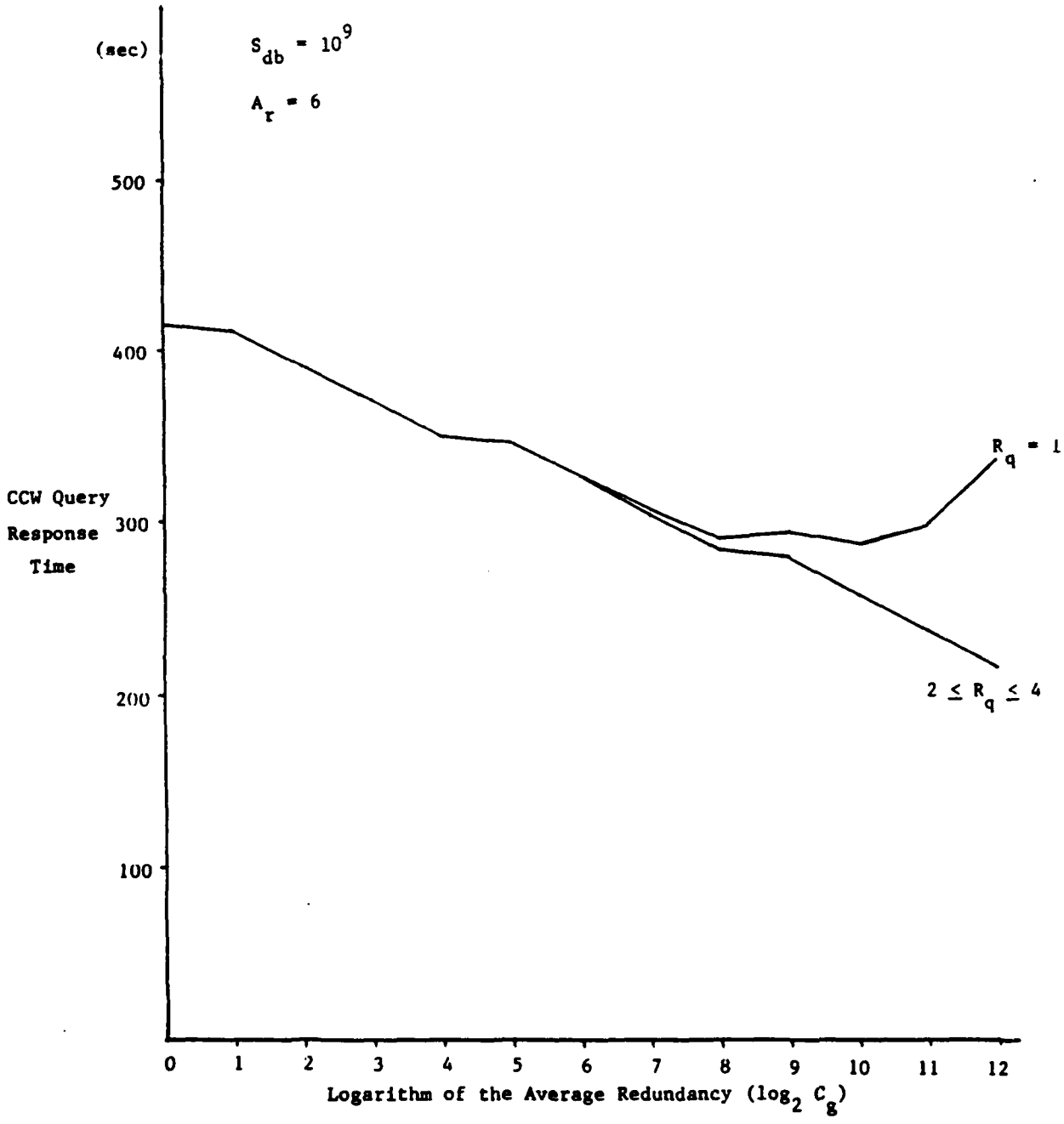


Figure 5.26 Effect of the Average Number of Arguments in a Query and the Average Redundancy on the CCW Query Response Time ($S_{db} = 10^9$ bytes)

depend on C_g while S_{ccw} decreases as C_g increases. But when R_q is 1, C_g becomes the number of good drops and the query response time starts increasing when C_g is larger than a certain value, because of the increased database access time.

To compare the retrieval performance of SCW and CCW techniques, we plot QT_{scw} and QT_{ccw} in Figures 5.27 through 5.29 for various parameter values. From those figures we can see that QT_{ccw} is smaller than QT_{scw} when R_q is small. There are two reasons supporting this discussion:

1) S_{ccw} is smaller than S_{scw} when R_q is small.
2) Since an ideal hashing function is assumed, the CCW technique does not produce any false drops but the SCW technique inherently produces false drops.

As we can see from Figures 5.8 and 5.11, about 70% of the query response time is used for the surrogate file searching when S_{db} is 10^9 bytes. Therefore, if we use multiple processors and/or associative memories to speed up the searching, we can reduce the query response time considerably. This is the topic of a later section.

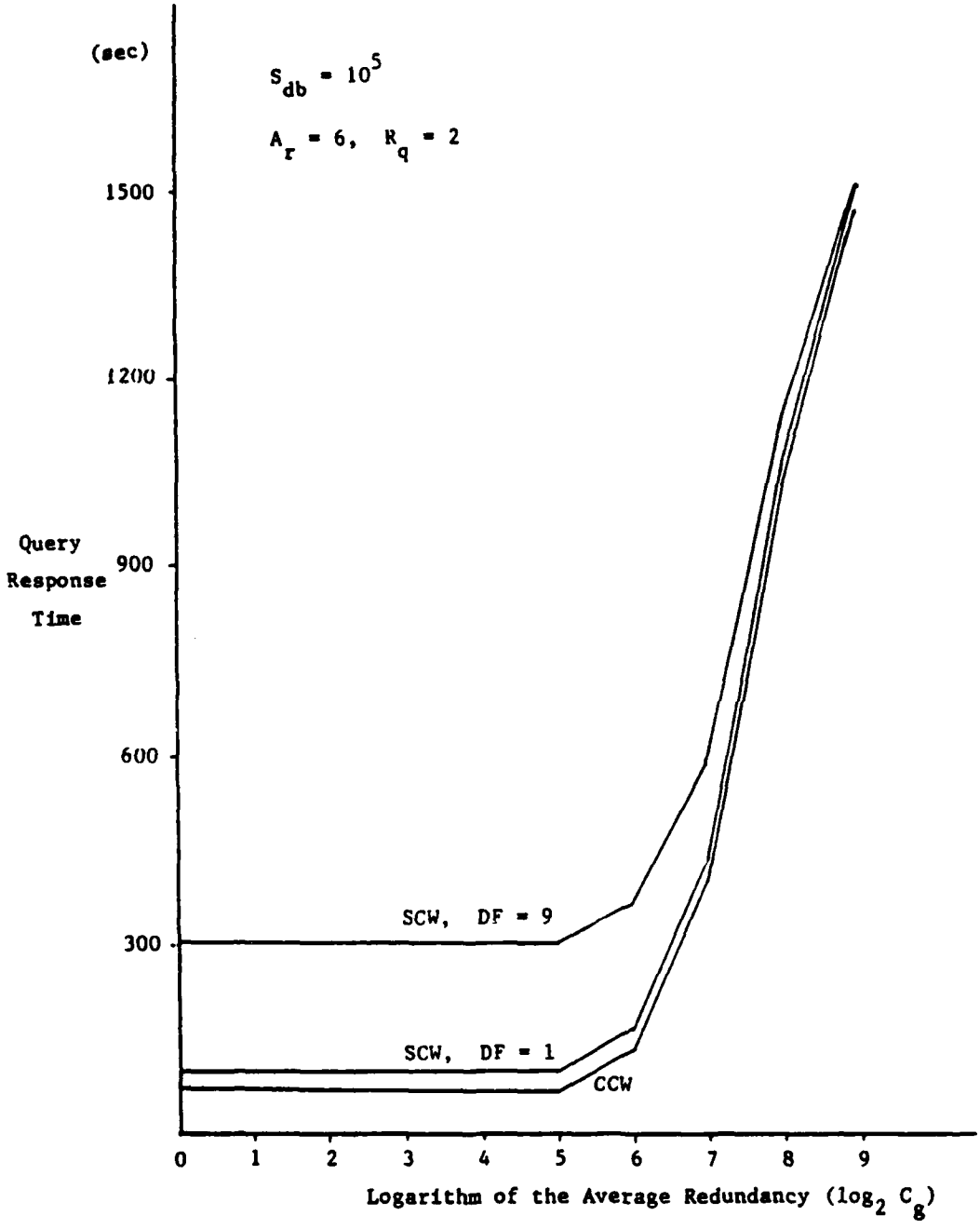


Figure 5.27 SCW and CCW Query Response Time Comparison ($S_{db} = 10^5$ bytes)

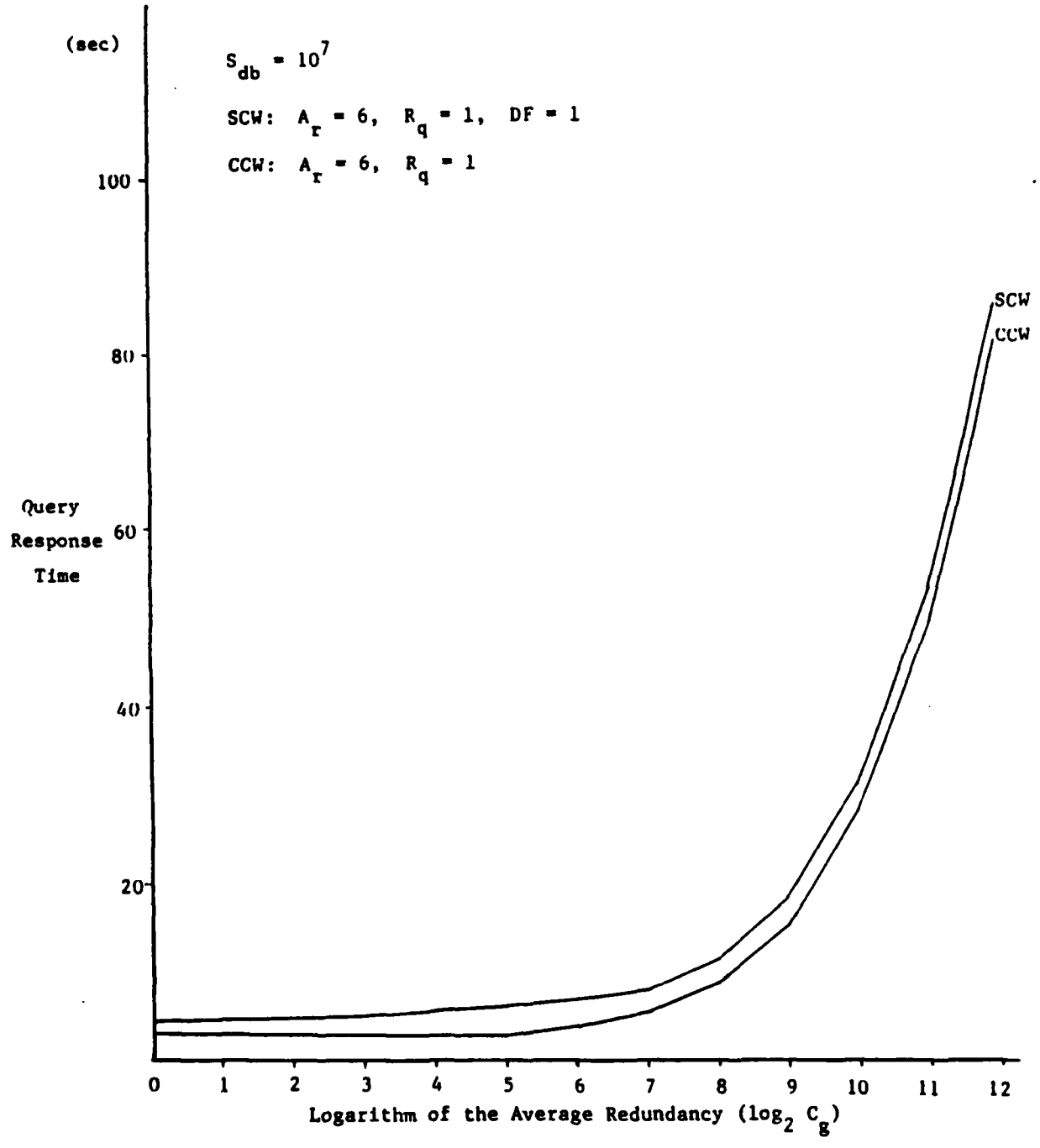


Figure 5.28 SCW and CCW Query Response Time Comparison ($S_{db} = 10^7$ bytes)

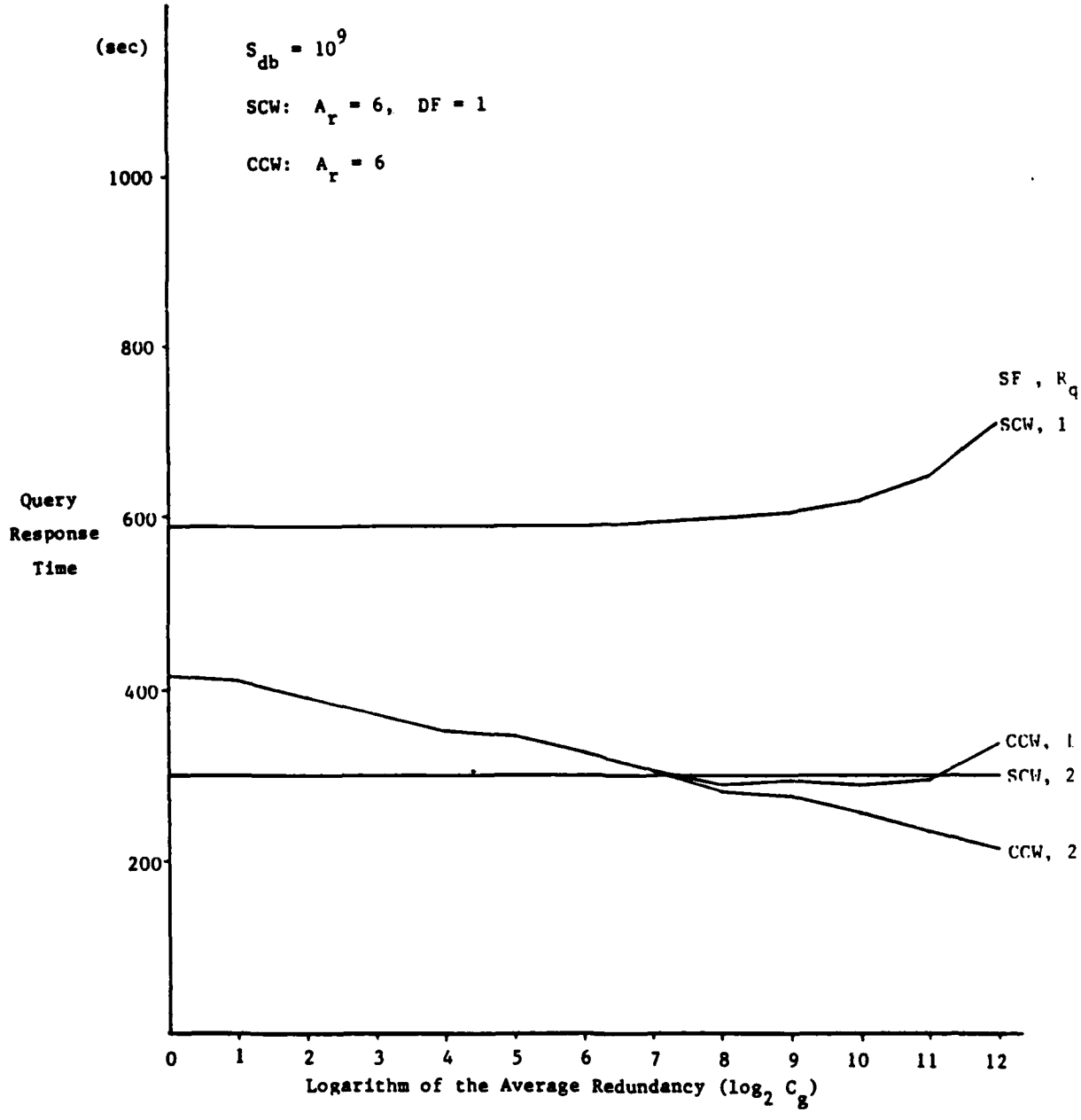


Figure 5.29 SCW and CCW Query Response Time Comparison ($S_{db} = 10^9$ bytes)

5.3. Maintenance Aspects of SCW and CCW Surrogate Files

It is very easy to update SCW or CCW surrogate files. When a new record is added to the database, the corresponding code word is simply appended to the existing SCW or CCW surrogate files. No other operations are required. Modification or deletion of an existing record requires a search for the entry in the surrogate file followed by the modification or deletion.

However, the partial match retrieval processing system using inverted list techniques has difficulties in maintaining index files although it can have fast retrieval capabilities. Whenever the record for a new fact is added the database, or an existing record is modified or deleted, all of index files must be retrieved and the reordering of indices could be required. The difficulties become more serious if the database grows fast or is frequently changed. To overcome the difficulties in maintenance with the inverted list technique, free space and an overflow area could be used in the index file. However, these approaches will increase the size of the index files so that query response time will increase.

Therefore, from a maintenance's point of view, SCW or CCW technique is suitable for the application requiring dynamic database, whose records are frequently updated.

6. System Model for Transformed Inverted Lists

Single or multi_level indexing is a common technique used in DBMS for fast data access. In partial match retrieval, creating index files for more than one field in a record is necessary. The extreme case arises when every entry in a record is indexed independently and is referred to as inverted lists organization. The problem behind using inverted lists is that the size of the indices becomes enormous, similar to or even larger than the database size.

Transformed Inverted Lists (TIL) are similar to inverted lists with the only difference that indices are built based on the binary representation (BR) of the hashed output of a given field in a record of the database relation. Two TIL types, TIL1 and TIL2, are considered in this study. A simple knowledge base system relation is illustrated in Figure 6.1. The fields are called arguments and the values for argument position 2 are listed.

6.1. TIL1 Description

TIL1 consists of a two level indexed inverted list. Figure 6.2 illustrates the TIL1 organization for argument position 2 of the relation shown in Figure 6.1. The blank entries are usually included for updating purposes. The secondary file for a given argument in a tuple is an ordered list of the BRs of the hashing function output of that argument with the attached unique identifier (Uid). The first entry in each block of this file is duplicated in the primary index file with an attached pointer to the corresponding secondary index block address. Furthermore, index files are partitioned in blocks of B bytes each. It is observed that the entries in the primary index file are ordered as well.

Uid	Ar_1	Ar_2	Ar_3	Ar_4
uid1	.....	br1	.....	.....
uid2		br2		
uid3		br3		
uid4		br1		
uid5		br4		
uid6		br1		
uid7		br5		
uid8		br6		
uid9		br6		
uid10		br7		
uid11		br3		
uid12		br4		

Figure 6.1 A Simple Knowledge Base Relation

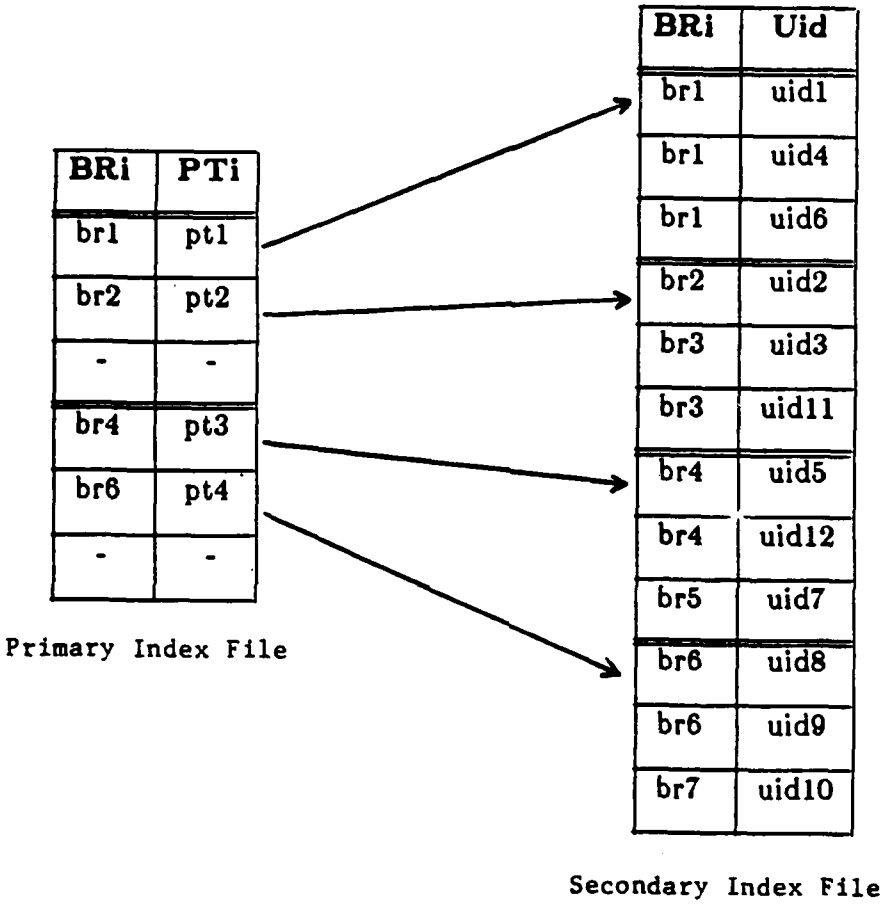


Figure 6.2 TIL1 for Ar_2 in Figure 6.1

When a given BR is to be retrieved (say BR=br3), the primary index file is sequentially accessed and searched with the pointer to the secondary block address corresponding to that BR retrieved (pt2 in our example). Then the secondary file is accessed in a direct mode and the required block retrieved and searched sequentially for the occurrence(s) of the requested BR. The output is a list of Uids (uid3 and uid11 for our example) corresponding to the value of the request.

6.2. TIL2 Description

TIL2 is a three level indexed inverted list organization and is illustrated in Figure 6.3 for the same example relation. The difference between TIL2 and TIL1 lies in that the TIL1 secondary index file is now split in two files: The TIL2 secondary index file and the tertiary index file. Each entry in the tertiary index file consists of a Uid, so that the number of entries in this file is equal to the number of records in the database relation. Each entry in the TIL2 secondary index file consists of three fields: the BR of the hashed function output of an argument value (say BR=br6), a list length entry "L" that provides the number of records in the database that have the same entry value in a given argument position (2 for br6) and a pointer to the address of the first Uid in the tertiary file that has BR=br6. This pointer consists of the block address and a displacement value in the block.

The retrieval process for TIL2 is similar to TIL1. The primary index file is first searched sequentially returning a pointer to a block in the secondary index file that contains the requested BR value. Then the secondary index file is accessed in a direct access mode and searched using the binary search technique to extract the respective list length and pointer address in the tertiary index file.

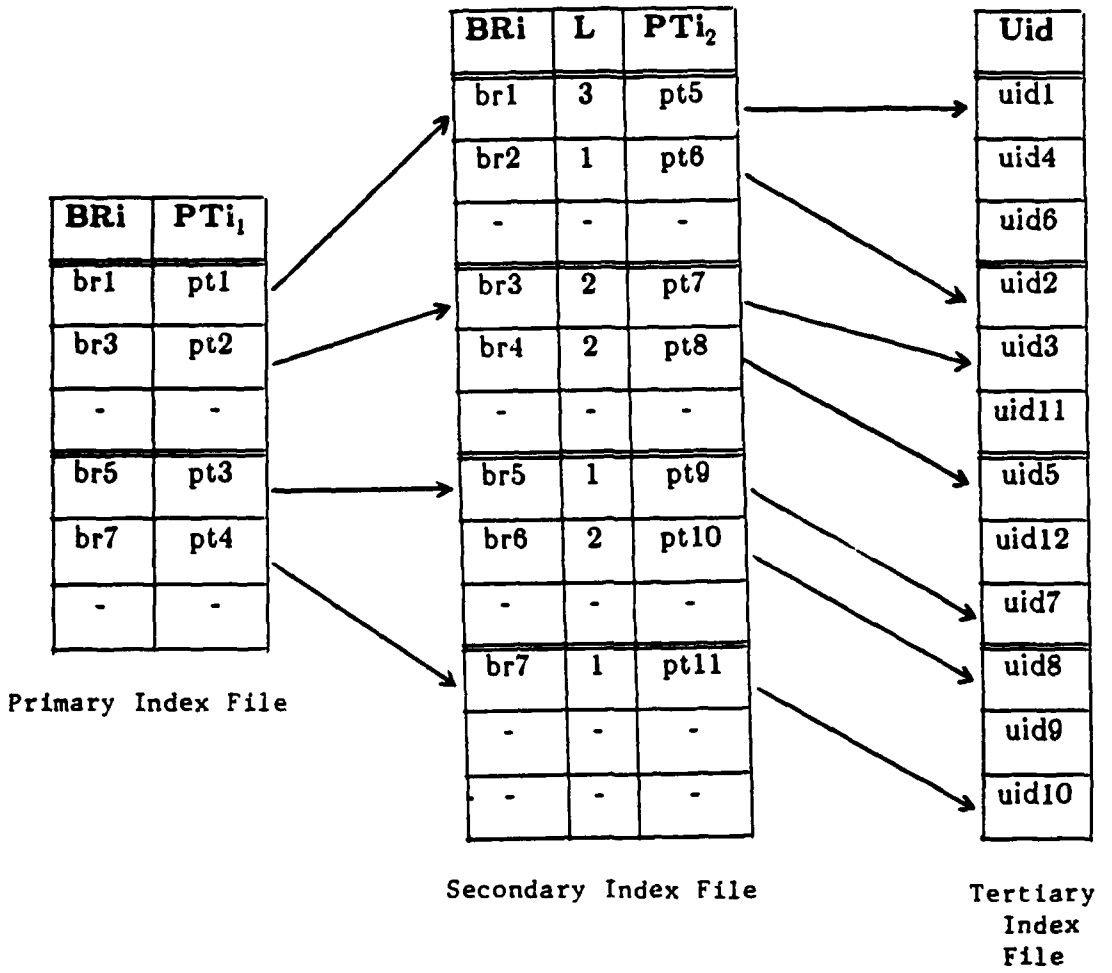


Figure 6.3 TIL2 for Ar_2 in Figure 6.1

Finally, the tertiary index file is accessed in direct access mode and a list of Uids of length L, is retrieved (uid8 and uid9 for br6).

6.3. Partial Match on Multiple Argument Positions

When more than one argument position match is requested in a query, the different outputs from the inverted lists searches need to be intersected. The outcome of the intersection is a set of Uids that complies with the query requirements. Finally this set of Uids is used to directly access the main database for the retrieval of the matched records.

It is noted that the gain in retrieval time when using transformed inverted lists is mainly due to the "channeled" approach inherent in indexed files. The data to be searched is in general much smaller than the relation size under consideration. Our subsequent analysis on TIL files relies on Cardenas (1975).

7. TIL Storage Requirements

In this section the minimal sizes of TIL files are derived assuming no blank entries are available in the index files. Those sizes are based upon the following parameters:

1) The bit length of the hashing function output for an argument in a record, denoted BR, must be at least $\left\lceil \log_2 \frac{N}{C_i} \right\rceil$ bits where C_i, called the value distribution factor, is the average number of records whose i-th arguments have the same value.

2) The Unique Identifier (Uid) for each tuple is encoded in $\lceil \log_2 N \rceil$ bits.

7.1. TIL1 Surrogate File Size

Denoting by S_{index1} and S_{index2} the minimal sizes of the primary and secondary index files of TIL1 respectively, the minimal size of TIL1 Surrogate Files is:

$$S_{TIL1} = S_{index2} + S_{index1} \tag{7.1}$$

7.1.1. Secondary Index File Size

Each entry in a secondary index file consists of two fields:

1) The Binary Representation (BR) of the hashing function output for an argument.

2) The Uid of the tuple in which it is found.

The number of entries in the secondary index file, for each argument, is equal to the number of tuples N in the database. Therefore, the secondary index size per argument position i (in bits) is given by:

$$S_{I2i} = \left(\left\lceil \log_2 \frac{N}{C_i} \right\rceil + \left\lceil \log_2 N \right\rceil \right) \times N \tag{7.2}$$

and the total secondary index file size is given by:

$$S_{index2} = \sum_{i=1}^{A_r} S_{I2i} \tag{7.3}$$

7.1.2. Primary Index File Size

Each entry in the primary index file consists of two fields: A pointer to a given block in the secondary index file, and the BR of the first argument value in that block.

The number of blocks in a secondary index file, denoted N_{bI2}, is equal to $\left\lceil \frac{S_{I2i}}{B} \right\rceil$, where B is the block size in bits/block. Each block address is thus encoded in $\left\lceil \log_2 N_{bI2} \right\rceil$ bits, and the number of entries in the primary index file of a given argument "i" is equal to the number of blocks of its secondary index file so that the total size of the primary index file is:

$$S_{index1} = \sum_{i=1}^{A_r} \left(\left(\left\lceil \log_2 \frac{N}{C_i} \right\rceil + \left\lceil \log_2 \left\lceil \frac{S_{I2i}}{B} \right\rceil \right\rceil \right) \times \left\lceil \frac{S_{I2i}}{B} \right\rceil \right) \tag{7.4}$$

7.2. TIL2 Surrogate File Size

Denoting by S_{index1}, S_{index2} and S_{index3} the respective sizes of the primary, secondary and tertiary index files for TIL2, the minimal size of TIL2 file is written as:

$$S_{TIL2} = S_{index3}+S_{index2}+S_{index1} \tag{7.5}$$

7.2.1. Tertiary Index File Size

An entry in the tertiary index file consists of the Uid pointing to the Database. The number of entries in this file is equal to the number of tuples N for each argument. The total number of entries is then $A_r \times N$ and the tertiary index file size is computed to be:

$$S_{index3} = \lceil \log_2 N \rceil \times A_r \times N \tag{7.6}$$

7.2.2. Secondary Index File Size

An entry in the Secondary Index file of an argument position i contains three fields:

1) A pointer to any possible record in the tertiary index file. This pointer is encoded with $\lceil \log_2 N \rceil$ bits.

2) A list length field (L), which is equal to the number of successive entries in the tertiary file that have the same argument values. This field is encoded with $\lceil \log_2 (C_i+1) \rceil$ bits. One is added to C_i to ensure a value of 1 bit for L when $C_i = 1$.

3) The BR of the output from the hashing function for a given argument value.

The number of entries in the secondary index file, in each argument position, is equal to $\left\lceil \frac{N}{C_i} \right\rceil$; that is the average number of distinct attribute values in each argument. The TIL2 secondary index file size per argument position i is then:

$$S_{I2i} = \left(\left\lceil \log_2 \frac{N}{C_i} \right\rceil + \left\lceil \log_2 (C_i+1) \right\rceil + \left\lceil \log_2 N \right\rceil \right) \times \left\lceil \frac{N}{C_i} \right\rceil \qquad (7.7)$$

and the total secondary index file size is:

$$S_{index2} = \sum_{i=1}^{A_r} S_{I2i} \qquad (7.8)$$

7.2.3. Primary Index File Size

Each primary index file entry contains two fields:

1) The BR as described for TIL1.

2) A pointer to the blocks in the secondary index file as described for TIL1. This pointer is encoded in $\left\lceil \log_2 N_{bi2} \right\rceil$, where N_{bi2} is the number of blocks in the secondary index file. The value for N_{bi2} is computed to be $\left\lceil \frac{S_{I2i}}{B} \right\rceil$.

The number of entries in the primary index file is equal to the number of blocks of the secondary index file so that the total size for TIL2 Primary Index file is given by:

$$S_{index1} = \sum_{i=1}^{A_r} \left(\left\lceil \log_2 \frac{N}{C_i} \right\rceil + \left\lceil \log_2 \left\lceil \frac{S_{I2i}}{B} \right\rceil \right\rceil \right) \times \left\lceil \frac{S_{I2i}}{B} \right\rceil \qquad (7.9)$$

8. Query Response Time of TIL

The query response time for TIL files depends on the following global parameters:

1) Database Relation Size: The surrogate file size is directly related to the size S_{db} of the database relation.

2) The method of obtaining the Uid pointers from the SF: It is assumed that binary search within each block and block skip retrieval are performed for secondary and primary index file access.

3) The number of arguments in a query, R_q.

4) The number of responses, or the selectivity factor of the query under consideration, that is dependent on the value distribution factor C_i and the number of tuples N in the relation.

In general, the retrieval process is divided into three major processes:

1) Surrogate File (SF) processing and Uid retrieval.

2) Uid intersection.

3) Database access to read the identified record(s) satisfying the query.

The different system characteristics definitions are:

1) The average block seek time, denoted by T_s.

2) The average block read time, denoted by T_r.

It is equal to $\frac{B}{\text{Transfer Rate}}$.

3) The average byte comparison time, denoted by T_B.

4) The average BR comparison time, denoted by T_c. It is equal to $\frac{BR}{8} \times \frac{T_B}{2}$.

8.1. TIL1 Query Response Time

In the case of TIL1, the major processes are subdivided into the following sub-processes:

1. Surrogate File Processing Time:
    1.1 Primary Index Retrieval time.
    1.2 Primary Index Search Time.
    1.3 Secondary Index Retrieval Time.
    1.4 Secondary Index Search Time.
2. Intersection Time for the different argument responses.
3. Database Access Time.

The TIL1 query response time (QT_{TIL1}) is equal to the sum of the SF processing time (T_{SF1}), intersection time (IT) and the database access time (DA_{TIL}):

$$QT_{TIL1} = T_{SF1} + IT + DA_{TIL} \tag{8.1}$$

8.1.1. Primary Index Retrieval Time

The following assumptions are made:

1) Each block in the primary index file is ordered internally with respect to the BR values and block skip search assumed for primary index file retrieval.

2) BRs are uniformly distributed over the blocks so that a given BR value is equally likely to be in any of the blocks of the primary index file.

3) All blocks reside on the same cylinder in secondary storage

Then the average number of blocks, N_{avgI1b}, to be retrieved for each query argument position is:

$$N_{avgI1b} = \frac{N_{I1b}+1}{2} = \frac{\left\lceil \frac{S_{I1i}}{B} \right\rceil + 1}{2} \tag{8.2}$$

With R_q as the number of arguments in a query, and sequential access of primary index blocks, the average TIL1 primary index retrieval time, $T_{index1-retrieval}$, is:

$$T_{index1-retrieval} = \sum_{i \in R_q} (T_s + T_r \times \frac{\left\lceil \frac{S_{I1i}}{B} \right\rceil + 1}{2}) \tag{8.3}$$

8.1.2. Primary Index Search Time

It is required to search one entry in each retrieved primary index block while the last retrieved block is *binary searched in* $\log_2 C_{I1bi}$ comparison steps; where C_{I1bi} is the number of entries in a primary index block and is equal to $\left\lfloor \frac{B}{BR + PTA} \right\rfloor$. The total primary index search time, $T_{index1-search}$, is therefore:

$$T_{index1-search} = \sum_{i \in R_q} T_c \times (N_{avgI1b} - 1 + \log_2 C_{I1bi}) \tag{8.4}$$

8.1.3. Secondary Index Retrieval Time

The TIL1 secondary index retrieval time is based on the average number of secondary index blocks, N_{avgib} to be retrieved for each argument position in a query. The total secondary index retrieval time, $T_{index2-retrieval}$, is given by:

$$T_{index2-retrieval} = \sum_{i \in R_q} (T_s + T_r \times N_{avgib}) \tag{8.5}$$

and with C_{I2bi} as the number of entries in a secondary index block, N_{avgib} is computed in the Appendix 1 and is as follows:

$$N_{avgib} = \frac{C_i}{C_{I2bi}} + \left\lceil \frac{C_i}{C_{I2bi}} \right\rceil - \left\lfloor \frac{C_i}{C_{I2bi}} \right\rfloor - \frac{1}{C_{I2bi}} + d \tag{8.6}$$

where d=1 when $\frac{C_i}{C_{I2bi}}$ is an integer, 0 otherwise.

8.1.4. Secondary Index Search Time

Sequential search is required to retrieve all Uids that satisfy the query. The time required to compare a matched BR is $2 \times T_c$. Denoting by C_{I2bi} the number of entries per secondary index block, it may or may not be necessary to compare all entries in a block so that the total secondary index search time is computed to be upper bounded by:

$$T_{index2\text{-}search} \leq 2 \times T_c \times C_{I2bi} \times \sum_{i \in R_q} N_{avgib} \tag{8.7}$$

It is noted that in the upcoming simulation section, the upper limit is assumed. The lack of precision in the above derivation does not affect the results as it will be shown later that block search time can be neglected.

8.1.5. Intersection Time

Two cases are considered:

$R_q = 1$: no intersection is required and the number of good drops (DG) is C_i.

$R_q > 1$: When $R_q = 2$, $C_i \times C_j$ comparisons are required and the number of good drops is given by $\frac{C_i \times C_j}{N}$. With $R_q = 3$, the number of required comparisons is now $C_i \times C_j + \frac{C_i \times C_j \times C_k}{N}$ and the number of good drops is $\frac{C_i \times C_j \times C_k}{N^2}$;

where $C_i \neq C_j \neq C_k$ in general. For a given R_q, the number of comparisons is given by:

$$\sum_{i=1}^{R_q-1} \frac{\prod_i C_i}{N^{i-1}}$$

and the number of good drops is given by:

$$DG = \begin{cases} N \prod_{i \in R_q} \left(\frac{C_i}{N}\right) & \text{if } DG > 1 \\ 1 & \text{otherwise} \end{cases} \tag{8.8}$$

The total intersection time is then written as:

$$IT = \begin{cases} \frac{T_{byte} \times \lceil \log_2 N \rceil}{2 \times 8} \times \sum_{i=1}^{R_q - 1} \frac{\prod_i C_i}{N^{i-1}} & \text{if } R_q > 1 \\ 0 & R_q = 1 \end{cases} \tag{8.9}$$

8.2. TIL2 Query Response Time

In the case of TIL2, the different subprocesses are subdivided as follows:

1. Surrogate File Processing Time:

    1.1 Primary Index Retrieval Time.

    1.2 Primary Index Search Time.

    1.3 Secondary Index Retrieval Time.

    1.4 Secondary Index Search Time.

    1.5 Access Time to the Tertiary File.

2. Intersection Time.

3. Database Access Time.

The TIL2 query response time is equal to the sum of the SF processing time T_{SF2}, intersection time IT and database access time DA_{TIL}:

$$QT_{TIL2} = T_{SF2} + IT + DA_{TIL} \tag{8.10}$$

8.2.1. Primary Index Retrieval Time

Similar to the TIL1 case, block skip search and sequential block retrieval are followed. The primary index retrieval time is given by:

$$T_{index1\text{-}retrieval} = \sum_{i \in R_q} (T_s + T_r \times N_{avgl1b}) \tag{8.11}$$

8.2.2. Primary Index Search Time

As for TIL1, binary search is performed within the last retrieved primary index block, while one comparison step is required for the remaining retrieved blocks. The primary index retrieval time is given by:

$$T_{index1\text{-}search} = \sum_{i \in R_q} T_c \times (N_{avl1b} - 1 + \log_2 C_{l1b}) \tag{8.12}$$

8.2.3. Secondary Index Retrieval Time

One disk access is required per query position i. Then the secondary index retrieval time is given by:

$$T_{index-retrieval2} = R_q \times T_b \tag{8.13}$$

8.2.4. Secondary Index Search Time

Binary search is performed due to the fact that the entries in an secondary index block are ordered with respect to the BR values. Then the secondary index search time is:

$$T_{index2-search} = R_q \times \log_2 C_b \times T_c \tag{8.14}$$

8.2.5. Tertiary File Access Time

The average number of blocks to be accessed is derived in a similar way as for the case of TIL1 secondary index retrieval and is given by:

$$N_{avgI3b} = \frac{C_i}{C_{I3bi}} + \left\lceil \frac{C_i}{C_{I3bi}} \right\rceil - \left\lfloor \frac{C_i}{C_{I3bi}} \right\rfloor - \frac{1}{C_{I3bi}} + d \tag{8.15}$$

The time required to retrieve the matched tuples is expressed by:

$$T_{index3} = T_s \times R_q + T_r \times \sum_{i \in R_q} N_{avgI3b} \tag{8.16}$$

8.2.6. Intersection Time

The TIL2 intersection time follows Equation 8.9, derived for the TIL1 intersection time in Section 8.1.5.

8.3. Database Access Time (TIL1 or TIL2)

From the analysis performed for the TIL1 intersection time in Section 8.1.5, the average number of good responses (DG) is expressed by Equation 8.8. Based on the probability ($\frac{1}{\left\lceil \frac{S_{db}}{B} \right\rceil}$) of a given response to be in a specific block, the database access time is derived as:

$$DA_{TIL} = T_b \times \left\lceil \frac{S_{db}}{B} \right\rceil \times \left(1-\left(1-\frac{1}{\left\lceil \frac{S_{db}}{B} \right\rceil}\right)^{DG}\right) \tag{8.17}$$

9. Simulation and Analysis of TIL Techniques

In the following analysis and computer evaluation for the derived TIL equations, the parameters are the same as the ones stated in Section 5 and are replicated here for clarity:

1) Each tuple in a database relation consists of a number of arguments, A_r, of 15 characters each. Denoting the size of the database by S_{db}, the number of tuples is:

$$N = \left\lfloor \frac{S_{db}}{15 \times A_r} \right\rfloor$$

2) In the derived equations, the C_i parameter is dependent on the argument position i. To simplify those equations, we have used an average value C_g:

$$C_g = \frac{\sum_{i=1}^{A_r} C_i}{A_r}$$

and similarly:

$$C_g' = \frac{\sum_{i \in R_q} C_i}{R_q}$$

then C_g and C_g' are independent from the argument position, and the derived equations are modified by replacing all sums over A_r or R_q by a multiplication term (A_r or R_q respectively). We shall assume $C_g = C_g'$ for further simplification.

3) The range of the variables under consideration are as follows:

S_{db} : 10^5, 10^7 and 10^9 bytes.

B : 2000 to 10000 bytes/block for a given S_{db}.

A_r : 2 to 10 for a given S_{db} and B.

C_g : Size Equations; 1 to 512.

Time Equations; 1 to 512 for $S_{db} = 10^5$, 1 to 4096 otherwise.

We observe that the upper limit of 4096 for C_g is not binding and that values in excess of 4096 could be expected in real situations. Nevertheless, we believe that our conclusions would not be affected drastically by this practical limitation and are precise for our purposes.

4) As given in Table 5.2, the system variables are

Seek Time (T_s): 28 msec.

Data Transfer Rate: 2000 bytes/msec.

Byte Comparison Time (T_B): 0.003 msec.

9.1. Transformed Inverted Lists Size

9.1.1. TIL1 Size

In Figure 9.1 %TIL1, the Surrogate File to Database size (S_{db}) ratio, is plotted versus the logarithm of the average redundancy factor ($\log_2 C_g$) for different S_{db} and A_r values. The following observations are made:

1) Effect of C_g: the curves are almost straight lines with a negative constant slope. This means that %TIL1 is proportional to $-\log_2 C_g$.

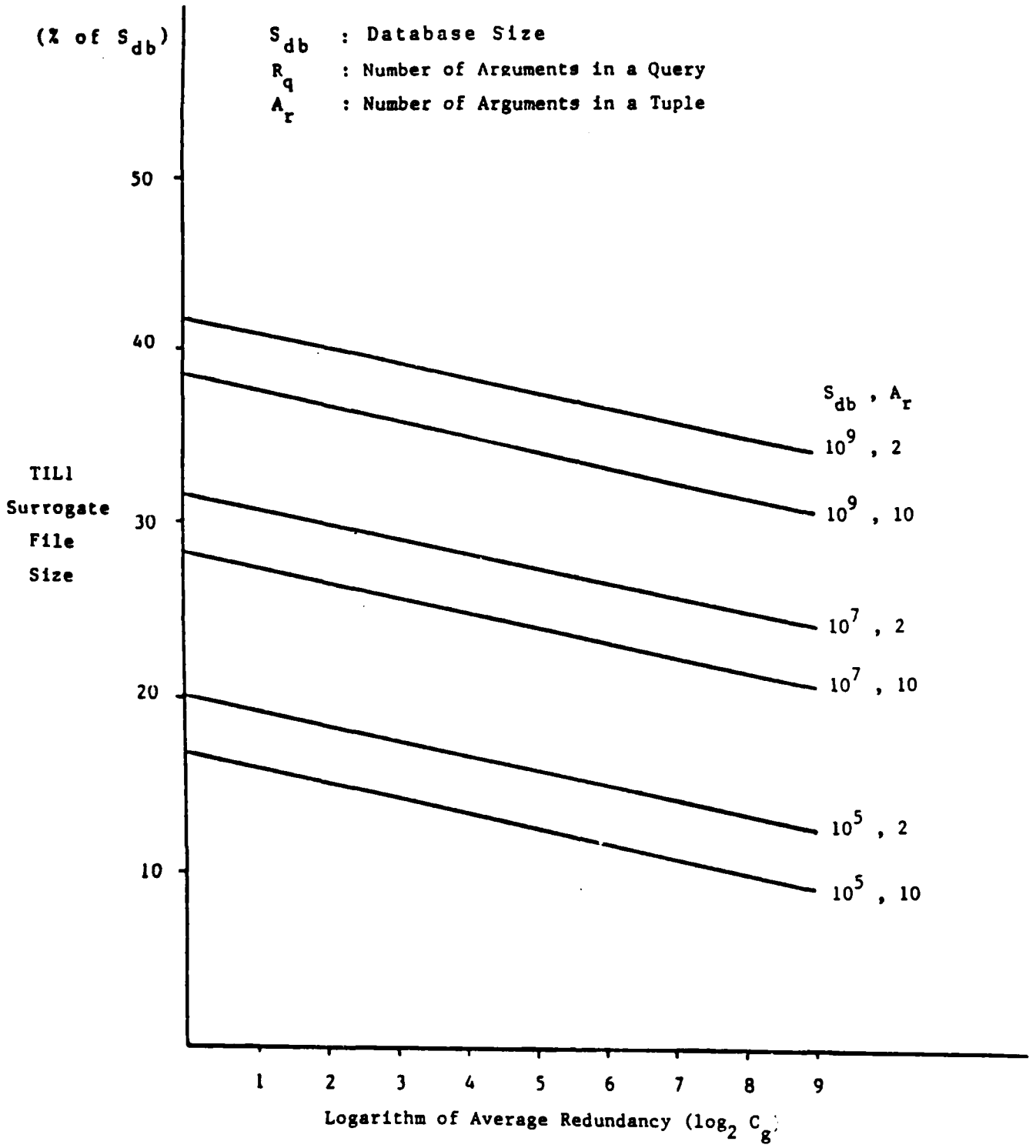


Figure 9.1 Effect of the DB Size and the Number of Arguments in a Tuple on the TIL1 Surrogate File Size.

2) Effect of the number of arguments A_r: for a given S_{db}, %TIL1 decreases with increasing values of A_r. This variation range is of the order of 5% for A_r varying from 2 to 10.

3) Effect of S_{db}: fixing A_r and C_g, %TIL1 increases with increasing S_{db}. This variation is detected to be proportional to $\log_2(S_{db})^2$.

4) Effect of the disk block size B: although not observed from Figure 9.1, our results showed that no practical change in surrogate file size is detected for the considered range of block size.

In general %TIl1 spans from a low of 9.2%, for $\log_2 C_g=9$, $A_r=10$ and $S_{db} = 10^5$, to 41.8% for $\log_2 C_g=0$, $A_r=2$ and $S_{db} = 10^9$. The above observation are justified by noting that $S_{index1} << S_{index2}$ in general, so that

$$S_{TIL1} \approx S_{index2} \approx A_r \times N \times \log_2 \frac{N^2}{C_g}$$

and

$$\%TIL1 \approx \frac{100}{15} \times \log_2 \frac{N^2}{C_g}$$

where $N = \left\lfloor \frac{S_{db}}{A_r \times 15} \right\rfloor$

9.1.2. TIL2 Size

The assumption that $S_{index1} << S_{index2}$ holds for TIL2 as it did for TIL1. %TIL2 is plotted versus $\log_2 C_g$ in Figure 9.2. The following observations are made:

1) For low values of C_g (i.e $\log_2 C_g<2$), %TIL2 is higher than %TIL1. This is due to the large size of the secondary index file. In this case the size of TIL2 secondary index file is similar to the TIL1 secondary index

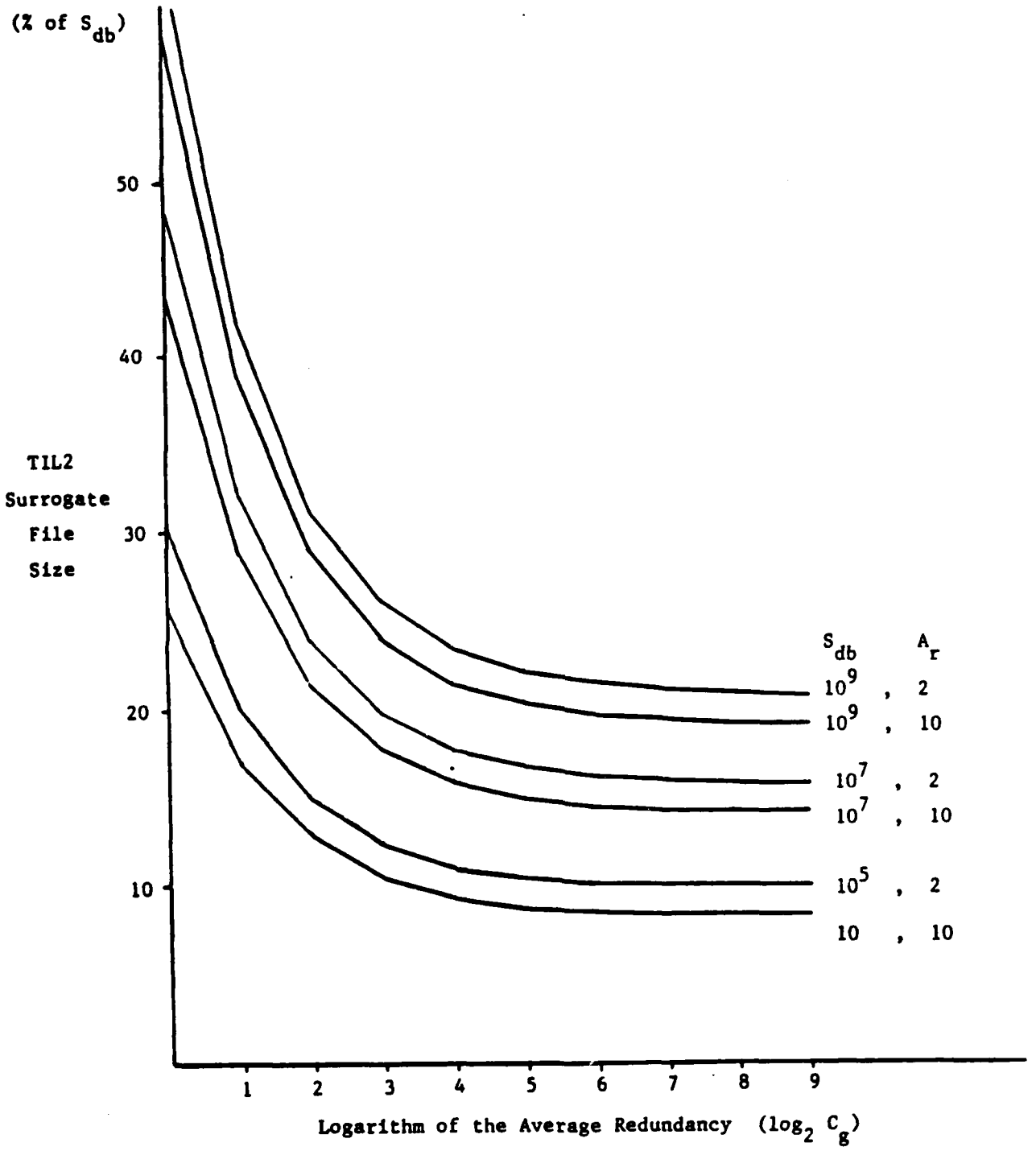


Figure 9.2 Effect of the DB Size and the Number of Arguments in a Tuple on the TIL2 Surrogate File Size.

file, so that the tertiary index file is somehow redundant and the global SF size of TIL2 is much larger than for TIL1. As C_g increases, TIL2 secondary index file size decreases sharply and tend to become negligible compared to the tertiary index file size. This accounts for the flatness of %TIL2 curves of Figure 9.2 for large C_g values. At this level, TIL2 size is approximated to be: $S_{TIL2} \approx S_{index3}$ for large C_g values.

2) Increasing A_r tends to decrease %TIL2 for a fixed S_{db} and C_g.

3) As for TIL1, our simulation results showed that %TIL2 is almost independent of the disk block size B.

Comparing Figures 9.1 and 9.2, it is noted that, for $\log_2 C_g > 3$, %TIL2 is less than %TIL1 and reaches a constant value for $\log_2 C_g > 9$.

9.2. Transformed Inverted Lists Response Time

The equations derived for the query response of TIL files are more complex than the one for minimum size. It is observed that index search times are negligible with respect to index files retrieval time. This is justified by $T_c << T_s$ or T_r. The general equation for the query response time of TIL techniques can be written:

$$T_{TIL} = T_{SF} + IT + DA_{TIL}$$

$$T_{SF} \approx \sum_{i} T_{indexi\_retrieval}$$

Depending on the size S_{db} of the database, we observe different effects on the query response time for the different system parameters and response time subentries.

9.2.1. Analysis of TIL1 Time Equations

9.2.1.1. Surrogate Files Retrieval Time

Three graphs are plotted in Figures 9.3, 9.4 and 9.5 for the index retrieval time of TIL1 versus $\log_2 C_i$. The following observations are made:

1) SF retrieval time increases with increasing values of R_q.

2) For $S_{db} = 10^5$ bytes (Figure 9.3), no dependency on C_g is detected. This is due to the small sizes of the files; only one block is to be accessed in all cases and this presents lower bounds on the retrieval process. In Figure 9.4, for $S_{db} = 10^7$ bytes, retrieval time is noticed to increase for values of $\log_2 C_g > 9$. This is due to the increasing number of sequential disk blocks that have to be read from the secondary index file. When S_{db} is increased further to 10^9 bytes, it is noted that retrieval time decreases first to a minimum, which is due to the decrease of the primary index file sizes, and increases again under the effect of the increasing number of disk block reads required to retrieve all secondary index file matches. The break point is around $\log_2 C_g = 9$ in Figure 9.5.

3) For small and medium sized relations, larger disk block sizes imply slower retrieval time as noticed in Figures 9.3 and 9.4. This is due to one disk block access in any case, so that disk block read time makes the difference. While for the large relation size of Figure 9.5, larger disk block sizes imply faster retrieval time and is due to a smaller number of required disk block reads.

Globally, index retrieval time is <750 msec for large R_q and even <350

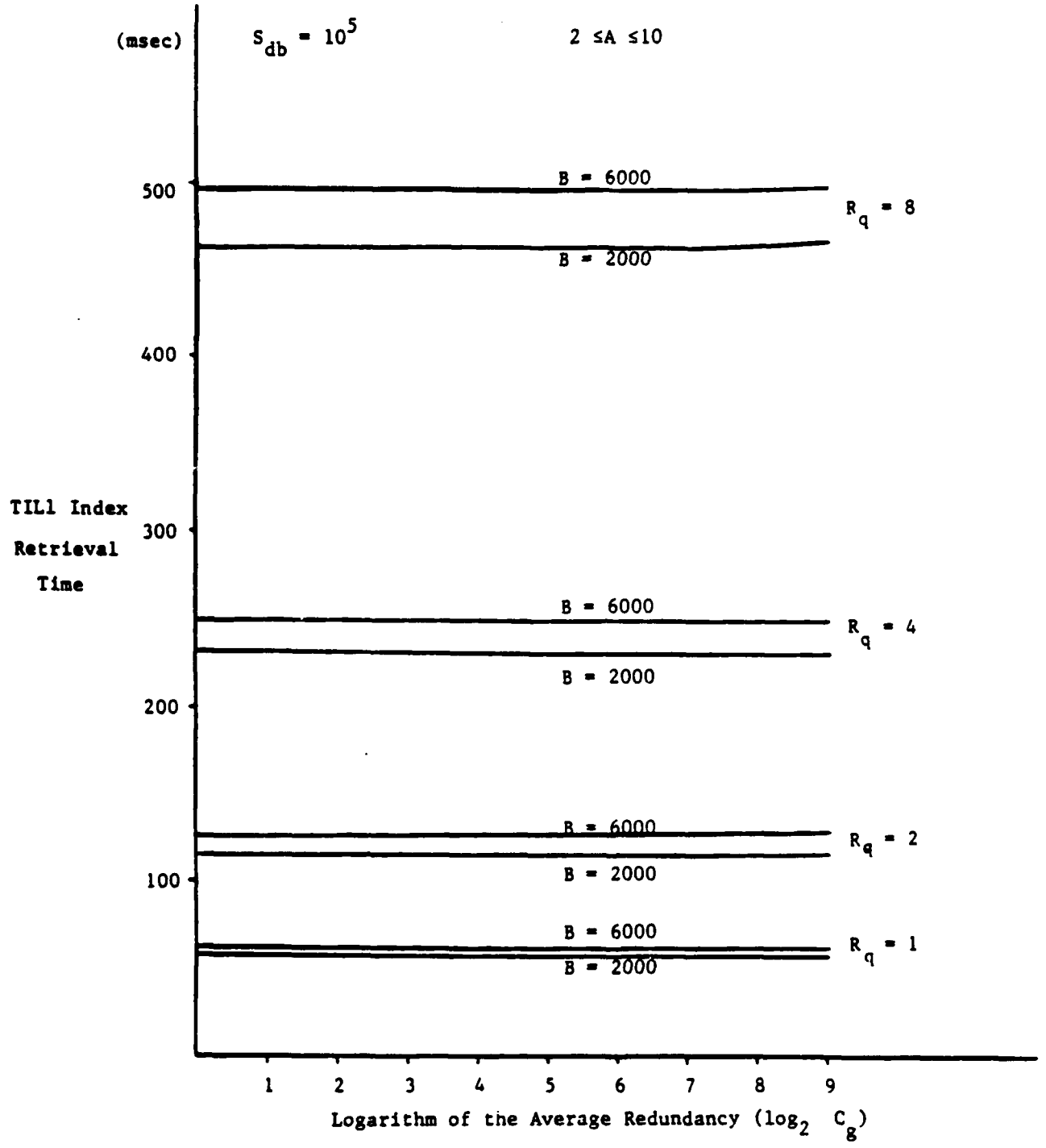


Figure 9.3 Effect of the Number of Arguments in a Query on TIL1 Index Retrieval Time ($S_{db} = 10^5$ bytes)

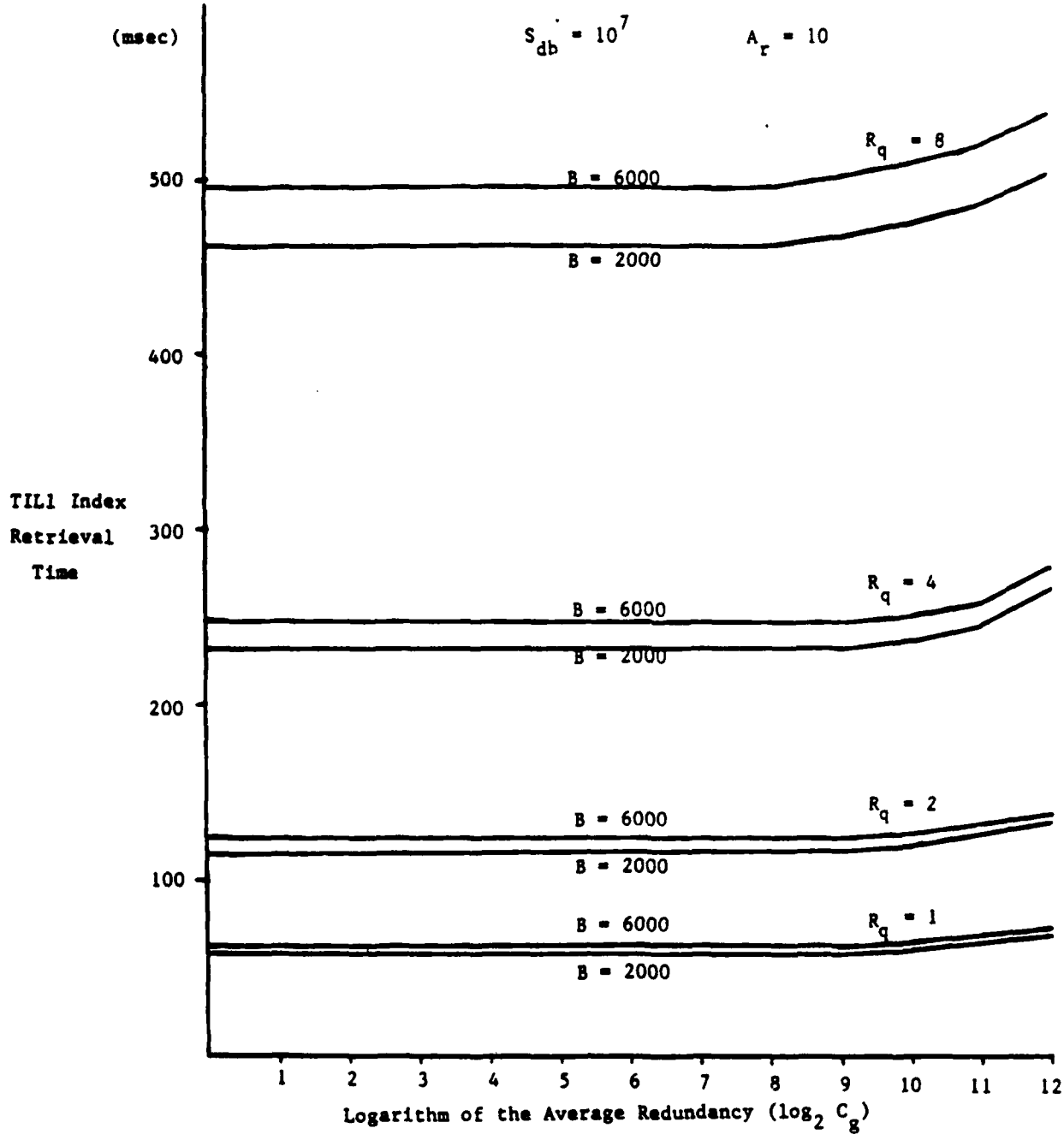


Figure 9.4 Effect of the Number of Arguments in a Query on TIL1 Index Retrieval Time ($S_{db} = 10^7$ bytes)

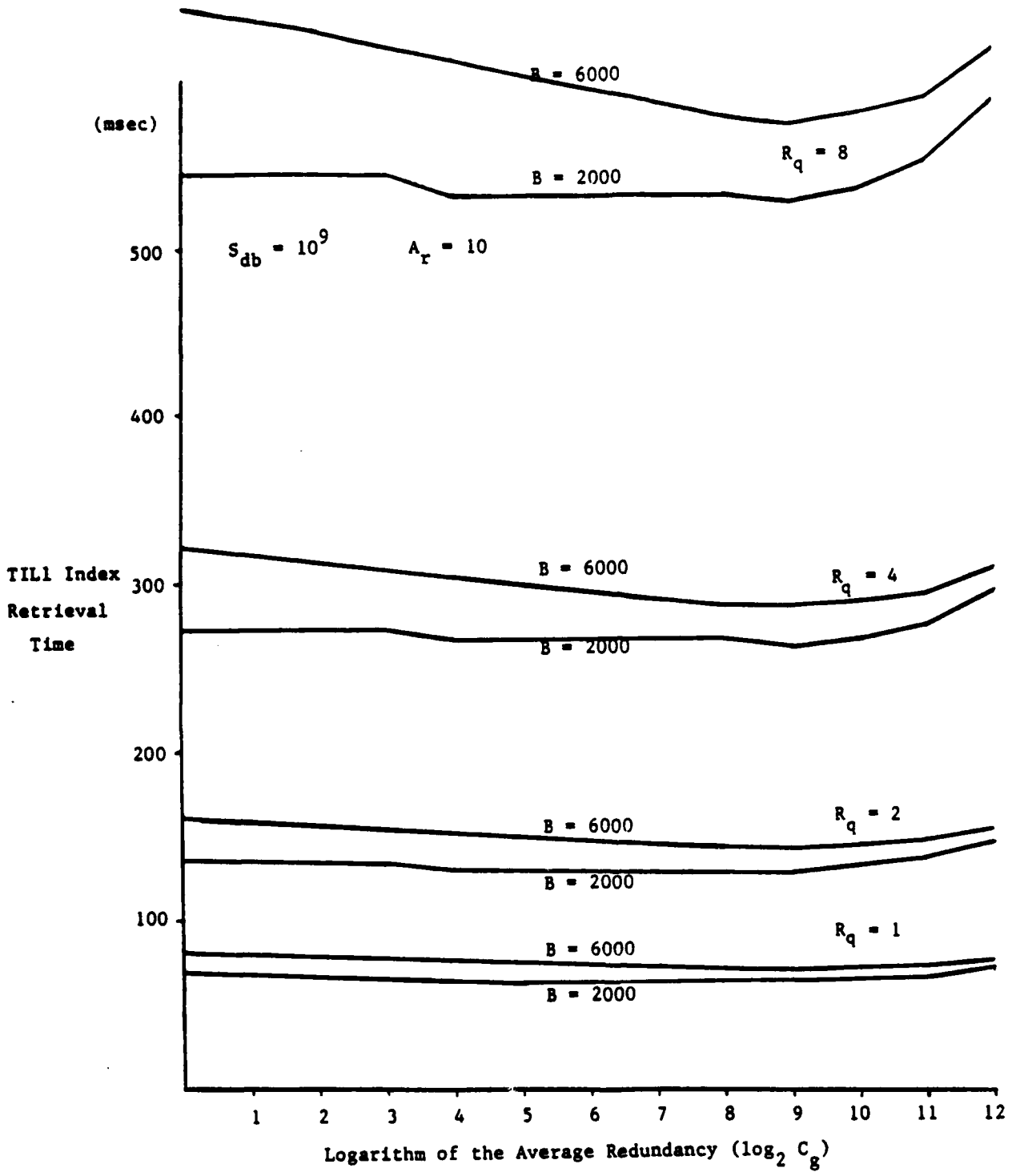


Figure 9.5 Effect of the Number of Arguments in a Query on TIL1 Index Retrieval Time (S_{db}= 10^9 bytes)

msec for medium R_q values.

9.2.1.2. Intersection Time

In this section we analyze intersection time bearing in mind that $R_q>1$; as for $R_q=1$, intersection time is 0. Figures 9.6 to 9.9 illustrate different plots of intersection time versus $\log_2 C_g$ for different parameters values. The mathematical basis that justifies those figures is as follows:

Intersection time has been derived to be:

$$IT = \frac{T_{byte}}{16} \times \lceil \log_2 N \rceil \times C_g^2 \times \sum_{i=0}^{R_q-2} \left(\frac{C_g}{N} \right)^i$$

If $\left(\frac{C_g}{N}\right)^{R_q} << 1$, then the above equation can be written as:

$$IT \sim \frac{T_{byte}}{16} \times \lceil \log_2 N \rceil \times C_g^2$$

It is also noted that intersection time is not affected by variations in disk block size B, but is highly dependent on C_g and the size S_{db} of the database. One observes that for large values of S_{db}, the convergence of the intersection equation is fast and thus for practical purposes the intersection time becomes independent from R_q.

Some observations on the graphs are:

1) For low values of C_g (i.e up to $\log_2 C_g = 9$), intersection time is acceptable for all considered database sizes and is less than 2 seconds (Figures 9.6 and 9.7). Those values are comparable to index retrieval time values.

2) For larger C_g values, the change in intersection time is fast so that values exceeding 25 seconds could be expected for large relation sizes

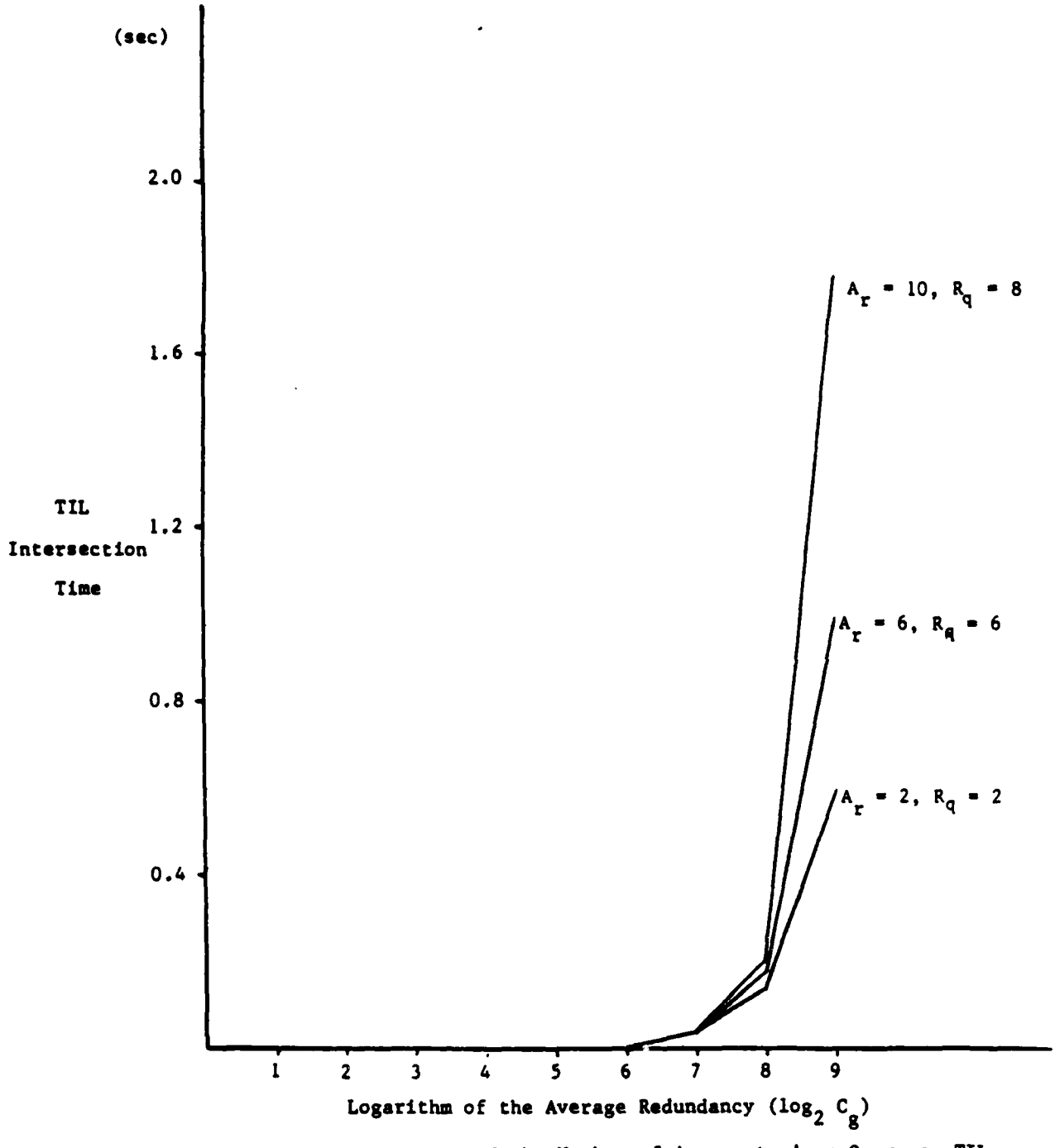


Figure 9.6 Effect of the Number of Arguments in a Query on TIL Intersection Time ($S_{db} = 10^5$ bytes)

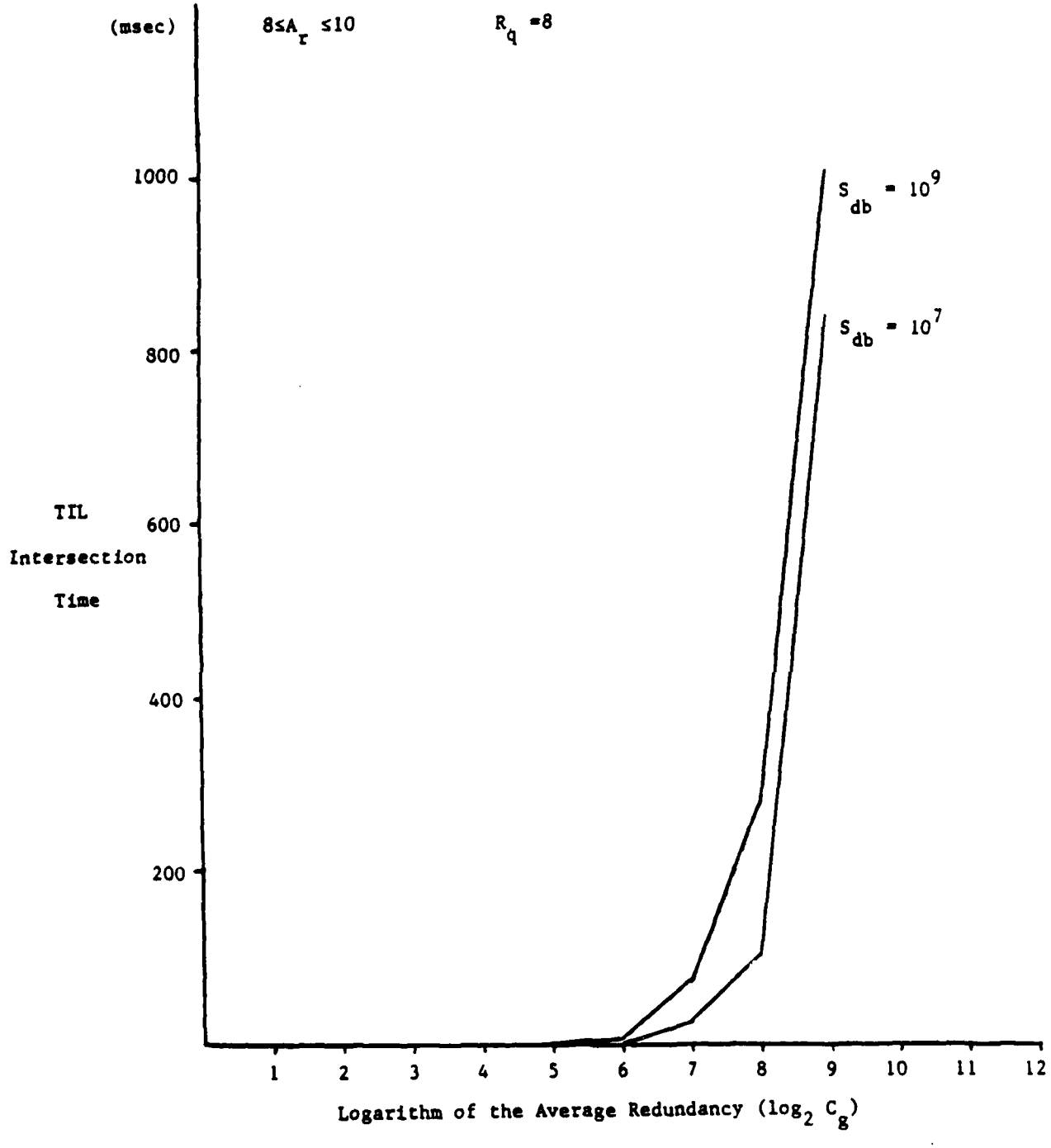


Figure 9.7 Comparison of TIL Intersection Time for Low Values of the Average Redundancy and Medium to Large DB Sizes.

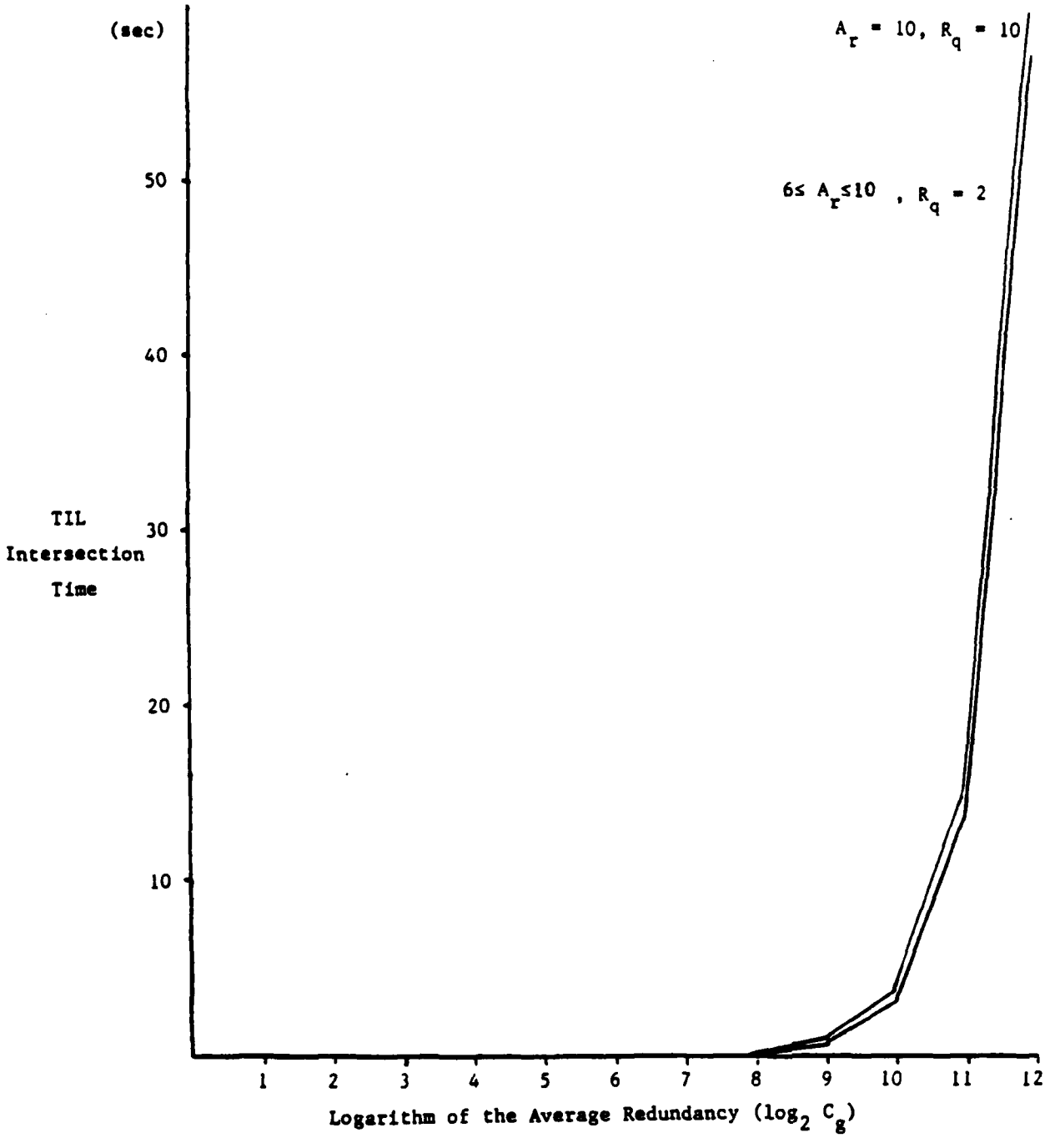


Figure 9.8 Effect of the Number of Arguments on the TIL Intersection Time (S_{db} = 10^7 bytes)

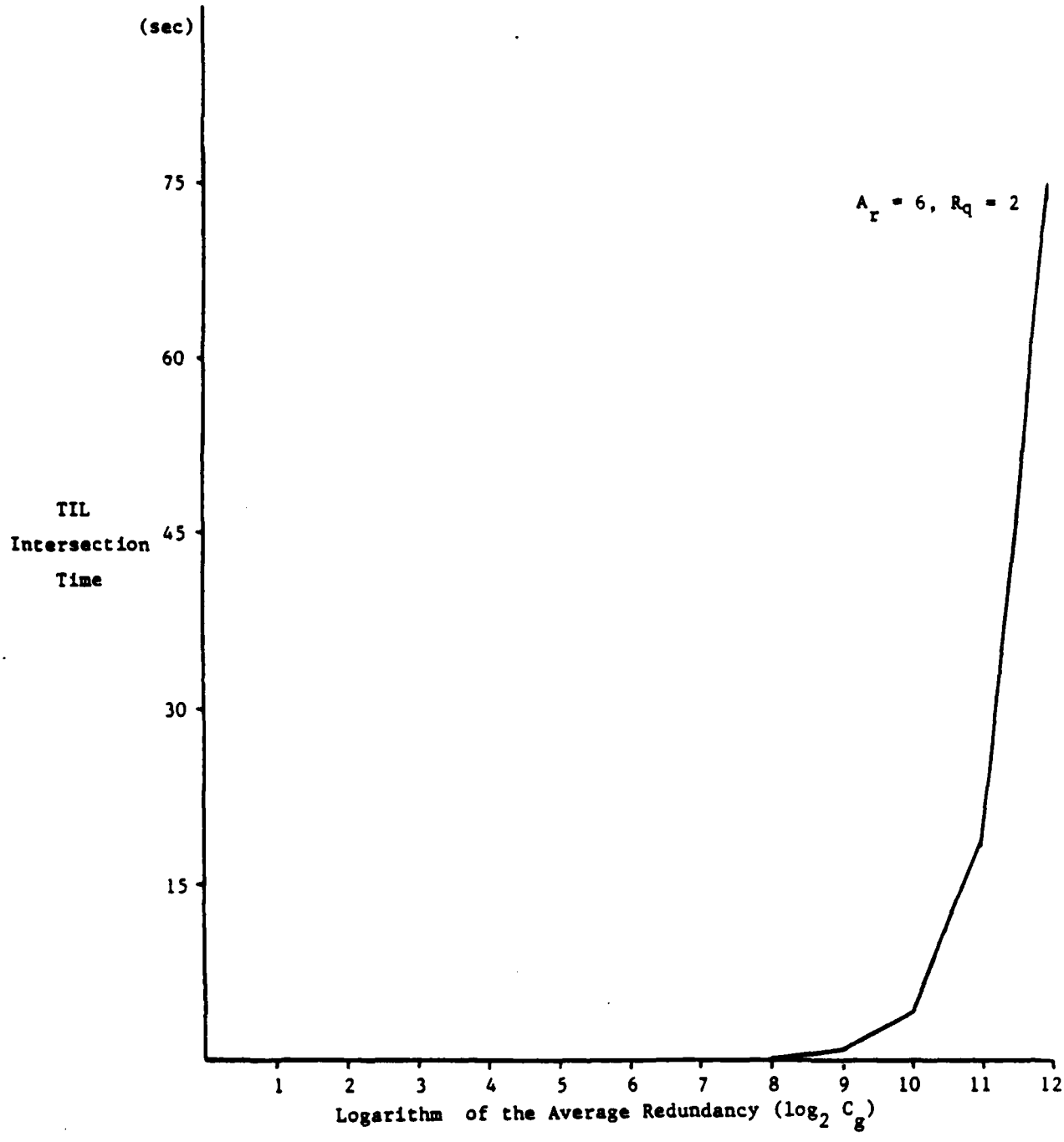


Figure 9.9 Effect of the Number of Arguments in a Query on the TIL Intersection Time ($S_{db} = 10^9$ bytes)

(Figures 9.8 and 9.9). For $\log_2 C_g = 12$ and relation sizes of 10^7 and 10^9, intersection time values greater than 50 and 70 seconds respectively are noticed.

3) The dependency on A_r and R_q is not critical for medium and large database sizes (Figures 9.8 and 9.9).

9.2.1.3. Database Access Time

For R_q=1, we observed that intersection time is 0. The overall query response time is then affected greatly by C_g as it determines the number of disk accesses to the main relation file. In this case, the equation for database access is given by:

$$DA_{TIL} = T_b \times \left\lceil \frac{S_{db}}{B} \right\rceil \times \left(1 - \left(1 - \frac{1}{\left\lceil \frac{S_{db}}{B} \right\rceil} \right)^{C_g} \right)$$

The overall shape of this equation is shown in Figure 9.10 for $S_{db} = 10^5$ bytes. The limit of this curve is $T_b \times \left\lceil \frac{S_{db}}{B} \right\rceil$, which accounts for the case of accessing the entire database. This limiting value is not practical since it can be shown, from a specific C_g value, that sequential access to the entire database, without resorting to the use of Surrogate Files would be faster. The break point being around $C_g = 32$ for $S_{db} = 10^5$ bytes but at a much larger C_g value for larger database sizes. This gives an upper bound for database access time to be $T_s + T_r \times \left\lceil \frac{S_{db}}{B} \right\rceil$. This break point is not considered further, since this value of C_g is not realistic for small database sizes. It is further noted that the ratio $\frac{C_g}{N}$ is of importance in determining the operating point for a given TIL based design;

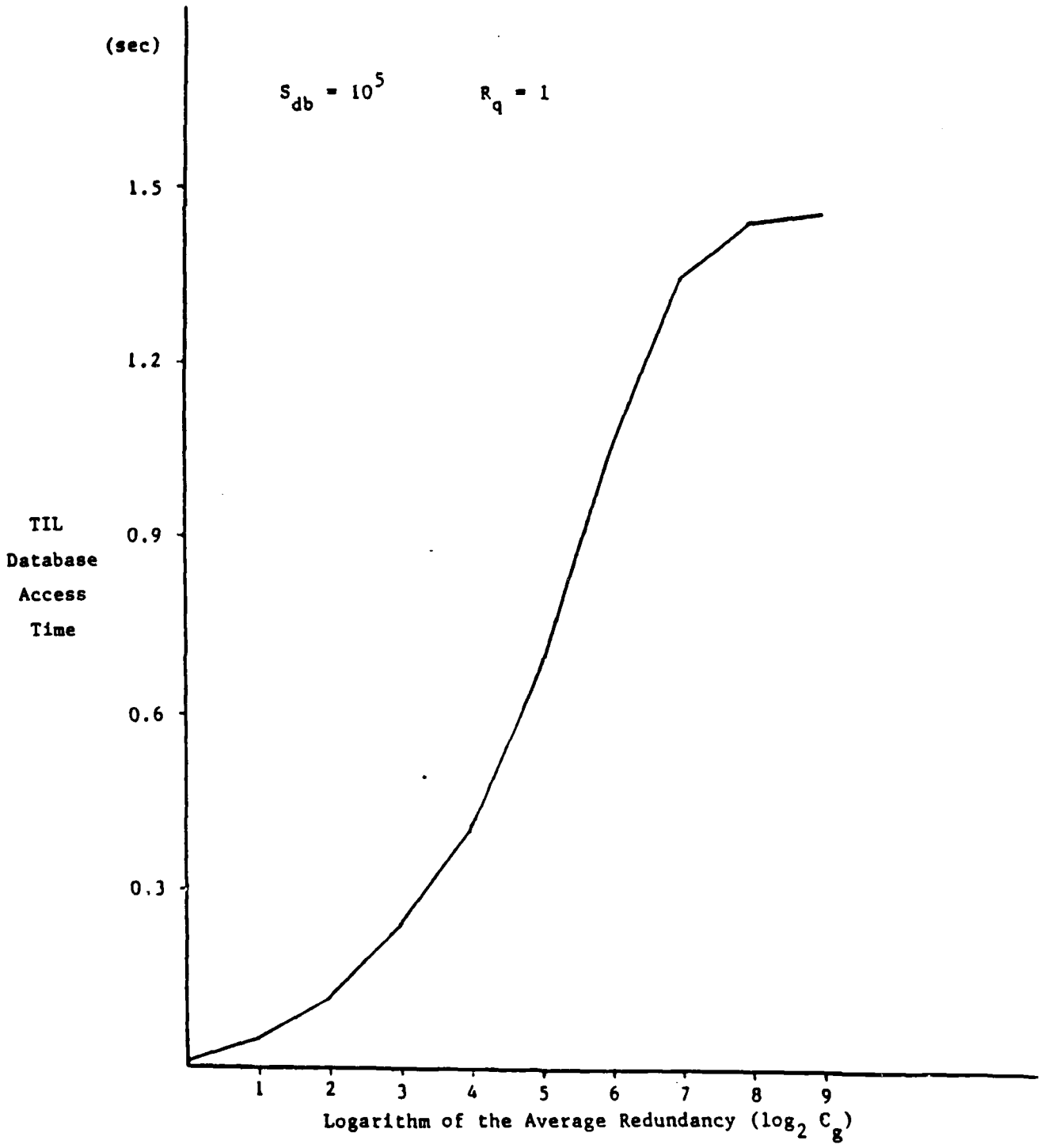


Figure 9.10 TIL Database Access Time ($S_{db} = 10^5$ bytes)

and as a general rule, TIL techniques are expected to be efficient for low values of $\frac{C_i}{N}$.

Figures 9.11 to 9.13 illustrate database access time for medium and large database sizes and $R_q = 1$. Index retrieval time is comparable to database access time for low values of C_g only, so that techniques to improve database access time response for large C_g values are necessary as high database response times are detected from Figures 9.11 and 9.12.

It is to be noted further that for $R_q > 1$, the number of good drops decreases with R_q, thus decreasing the number of disk accesses to the main database. With realistic values for $\frac{C_g}{N}$, main database access time becomes negligible with respect to intersection time or even index retrieval time in some cases. As has been shown in the previous section, intersection time plays a major role in determining the total query response time with respect to the TIL1 technique in those cases, but extending the analysis to even larger values of C_g (greater than 4096) would imply that there is a point where database response becomes of importance.

9.2.1.4. TIL1 Query Response Time

The Query Response Time for TIL1 technique is presented in Figures 9.14 to 9.20. In Figures 9.14, 9.15, 9.16, 9.19 and 9.20 Query Response Time and its corresponding subprocessing times are plotted for different database sizes and number of arguments in a query.

1) In Figure 9.14, $S_{db} = 10^5$ bytes and $R_q = 1$. It is noted that the normal operating point for the efficient use of TIL1 technique should be for

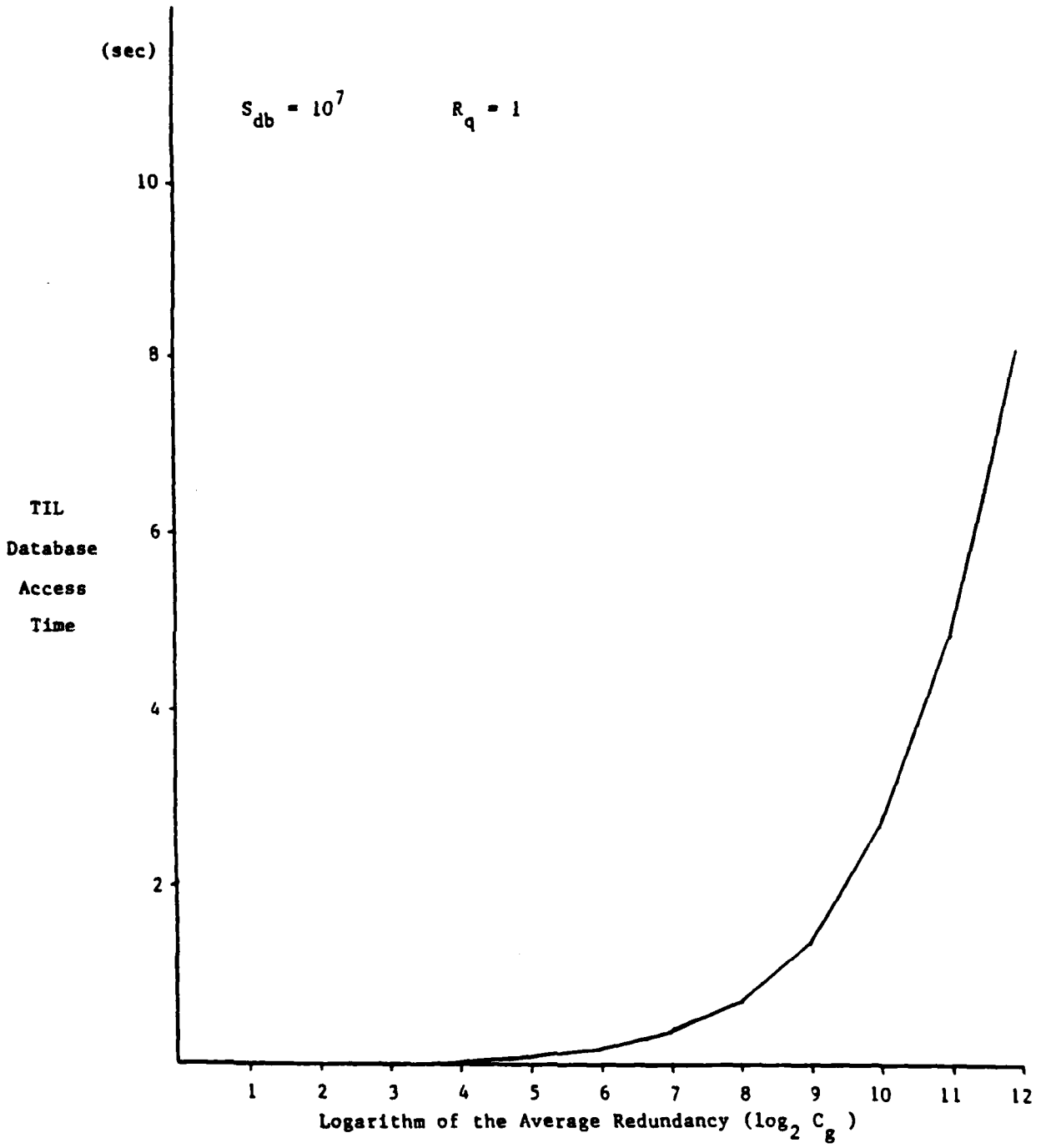


Figure 9.11 TIL Database Access Time ($S_{db} = 10^7$ bytes)

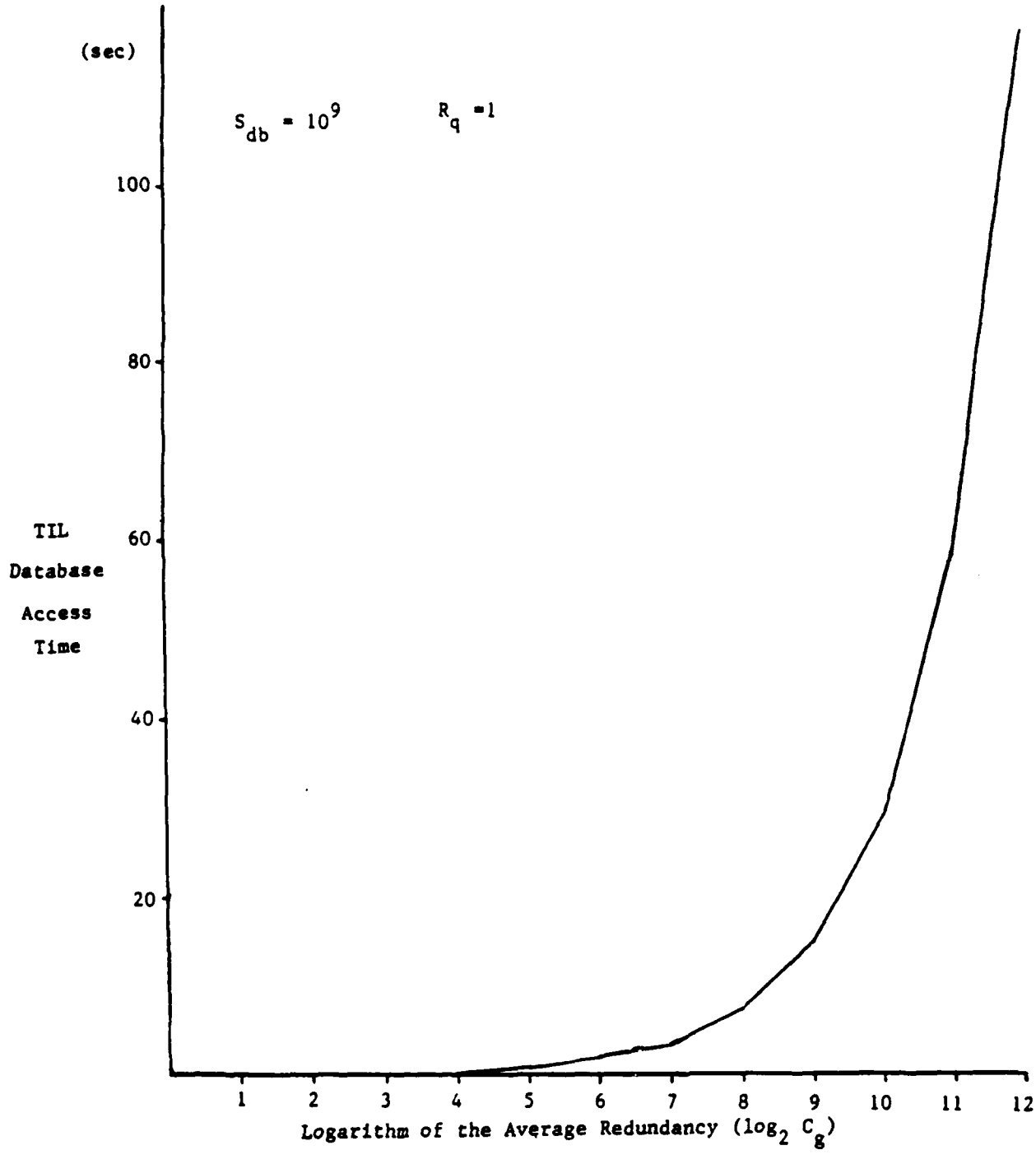


Figure 9.12 TIL Database Access Time ($S_{db} = 10^9$ bytes)

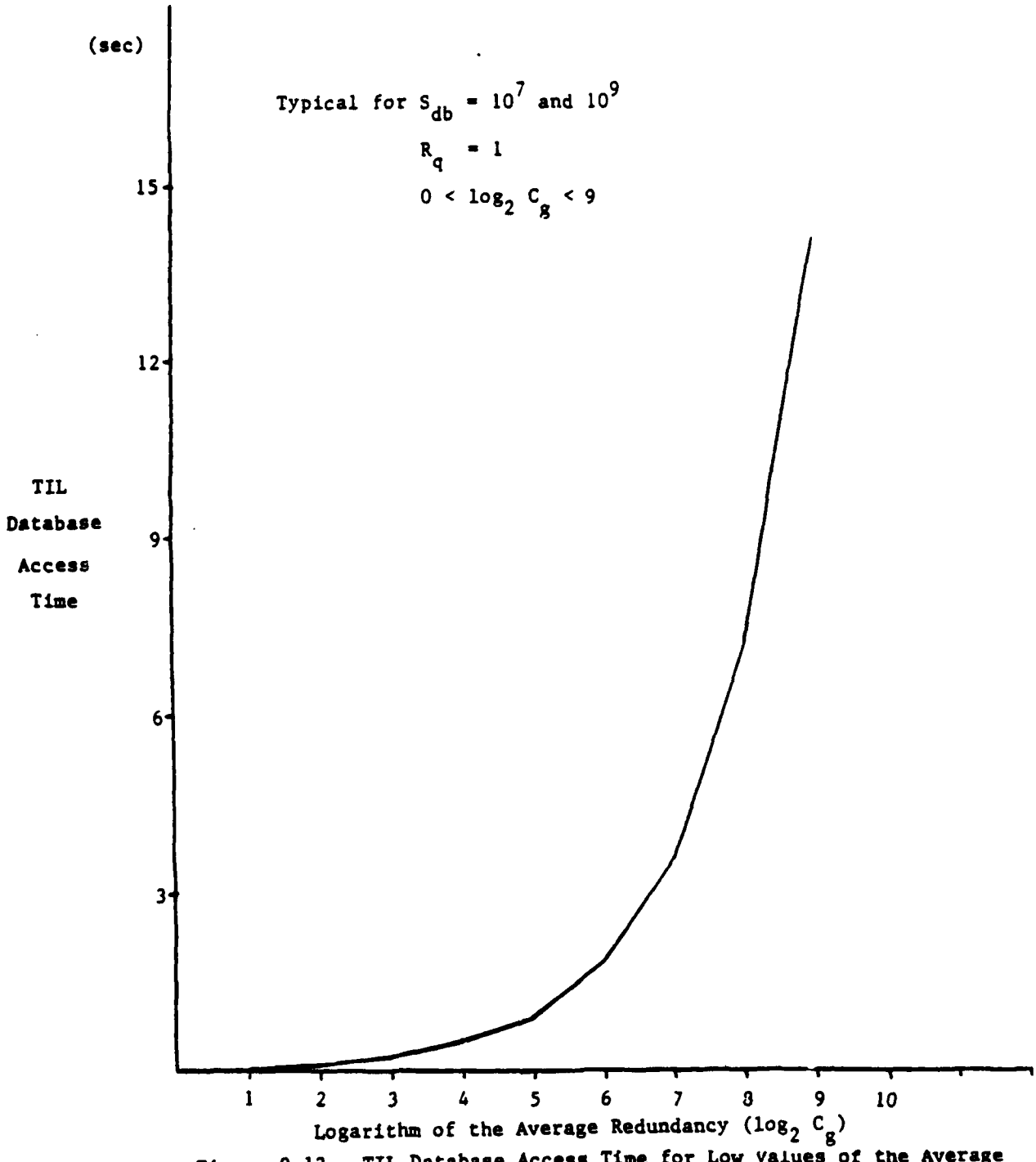


Figure 9.13 TIL Database Access Time for Low values of the Average Redundancy and Medium to Large DB Sizes.

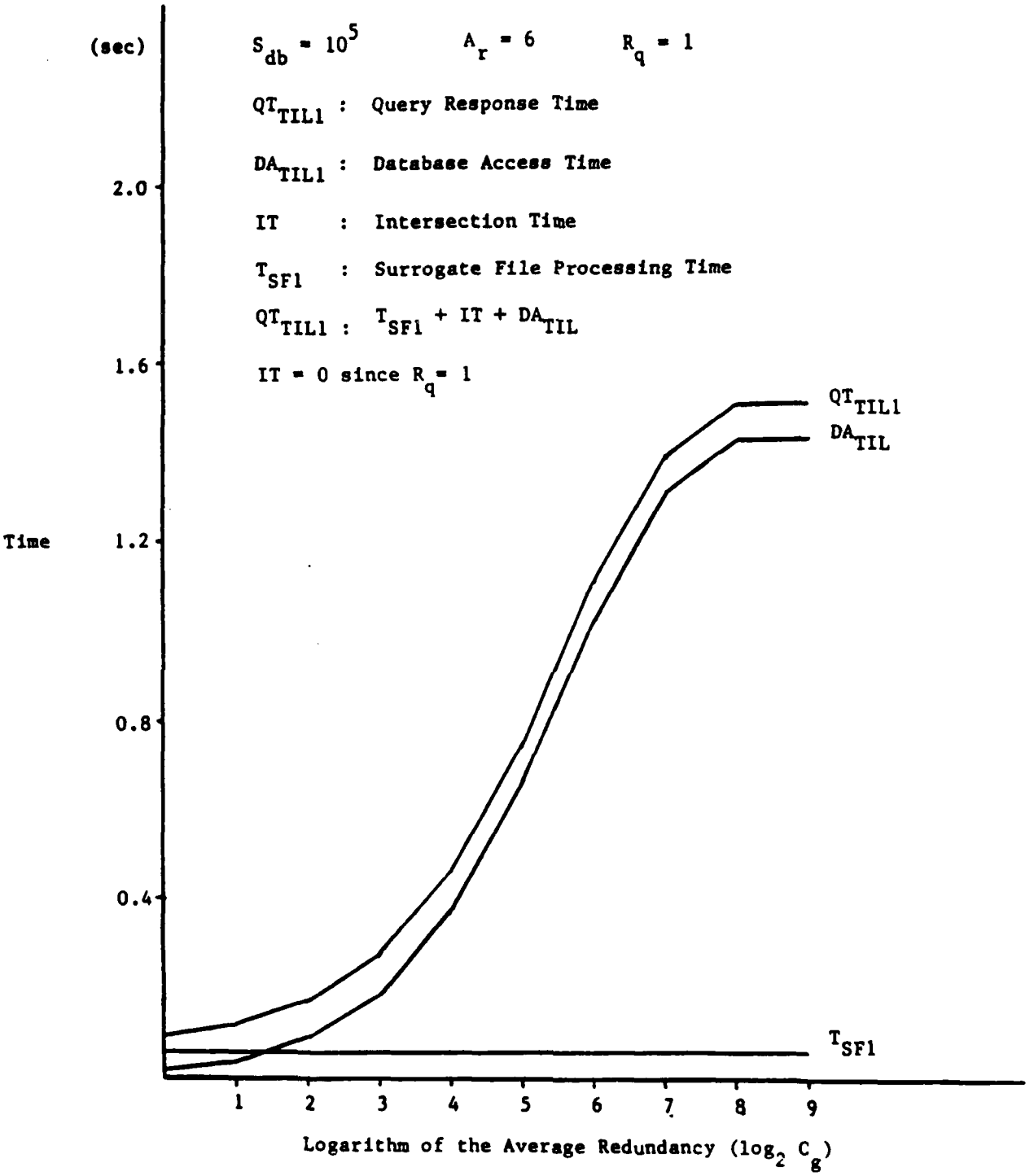


Figure 9.14 Components of the TIL1 Query Response Time ($S_{db} = 10^5$ bytes, $R_q = 1$)

values of $C_g < 32$ (i.e $\log_2 C_g = 5$). This is the inflection point on the query response time curve. To the right of this point, sequential search of the database becomes more efficient. From Figure 9.14, it is noticed that database access time is the dominant factor for $R_q = 1$.

2) Figures 9.15 and 9.16 illustrate the case when $R_q > 1$ and $S_{db} = 10^5$ bytes. Looking to the left of the $\log_2 C_g = 5$ point, Index retrieval time is found to be the dominant factor in the query response time. The right side of those plots is interesting in that it gives a feeling of what to expect for medium and large database sizes in case the value distribution factor C_g is very large.

3) In Figure 9.17, $R_q = 1$ and the query response time is plotted for $S_{db} = 10^7$ and 10^9 bytes. Those curves reflect the dependency of the query response on database access time. Values up to 80 and 120 seconds are observed for the respective database sizes.

4) In Figure 9.18, $R_q > 1$ and two plots for $S_{db} = 10^7$ and 10^9 bytes are shown. For $\log_2 C_g = 12$, query response times of 60 and 80 seconds respectively are observed.

5) Figures 9.19 and 9.20 are plotted to show the dependencies of query response time for medium and large database sizes with $R_q > 1$. It is observed that, for $\log_2 C_g < 8$, Index retrieval time is the dominant factor, while for $\log_2 C_g > 9$, intersection time becomes the essential element in determining the query response time for TIL1.

To summarize, for medium and large database relation sizes, it can be stated that:

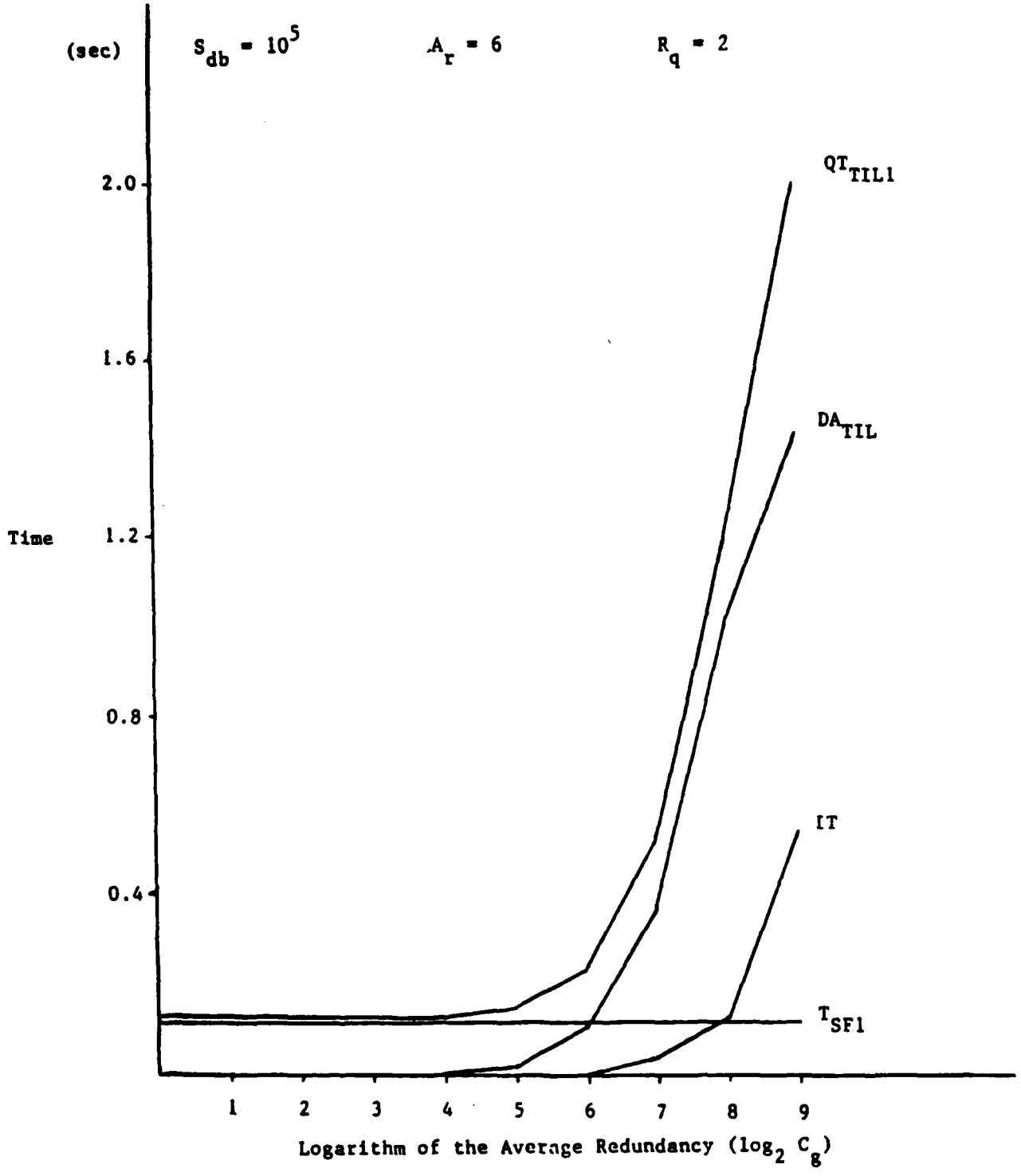


Figure 9.15 Components of the TIL1 Query Response Time ($S_{db} = 10^5$ bytes, $A_r = 6$, $R_q = 2$)

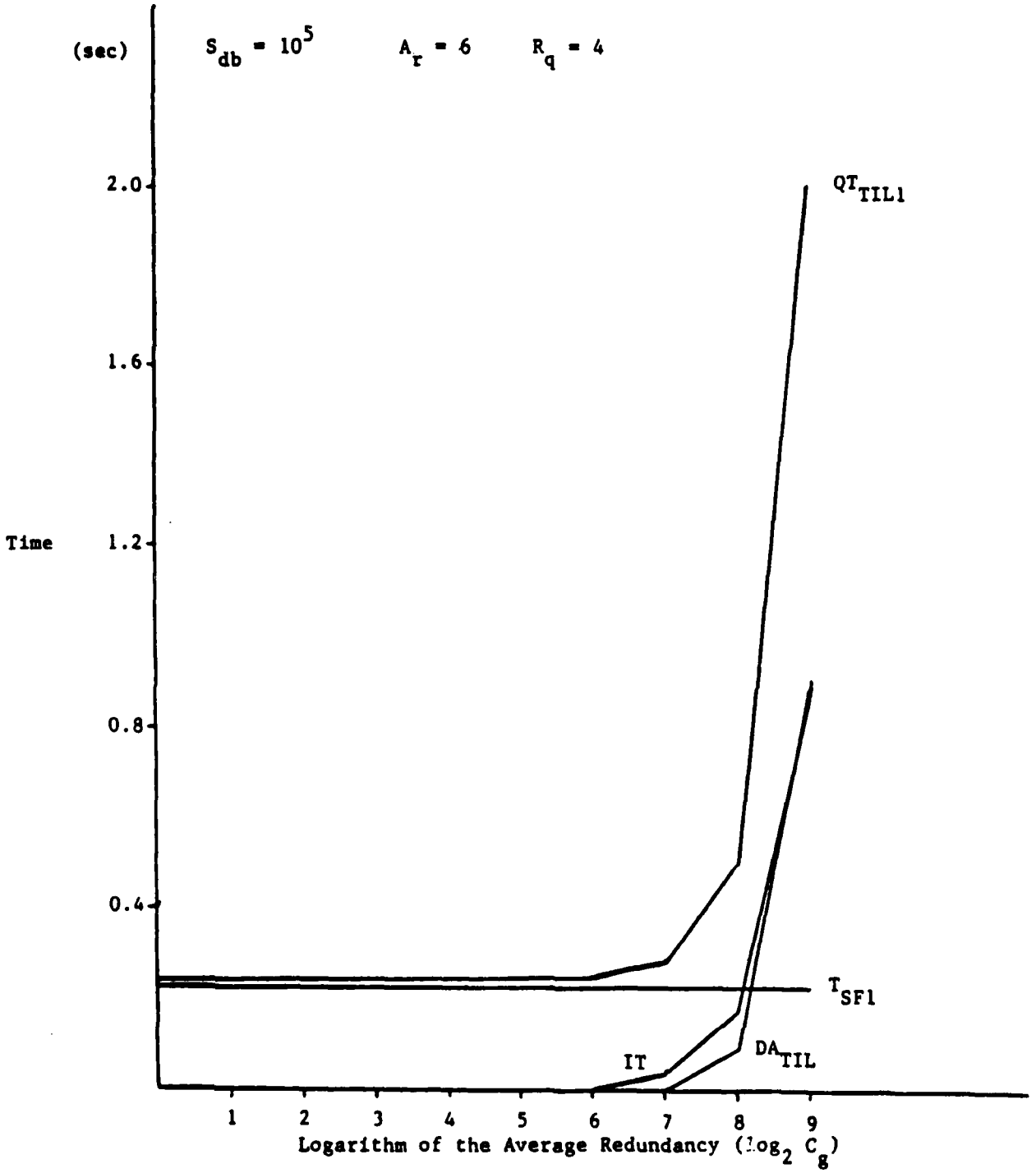


Figure 9.16 Components of the TIL1 Query Response Time ($S_{db} = 10^5$ bytes, $A_r = 6$, $R_q = 4$)

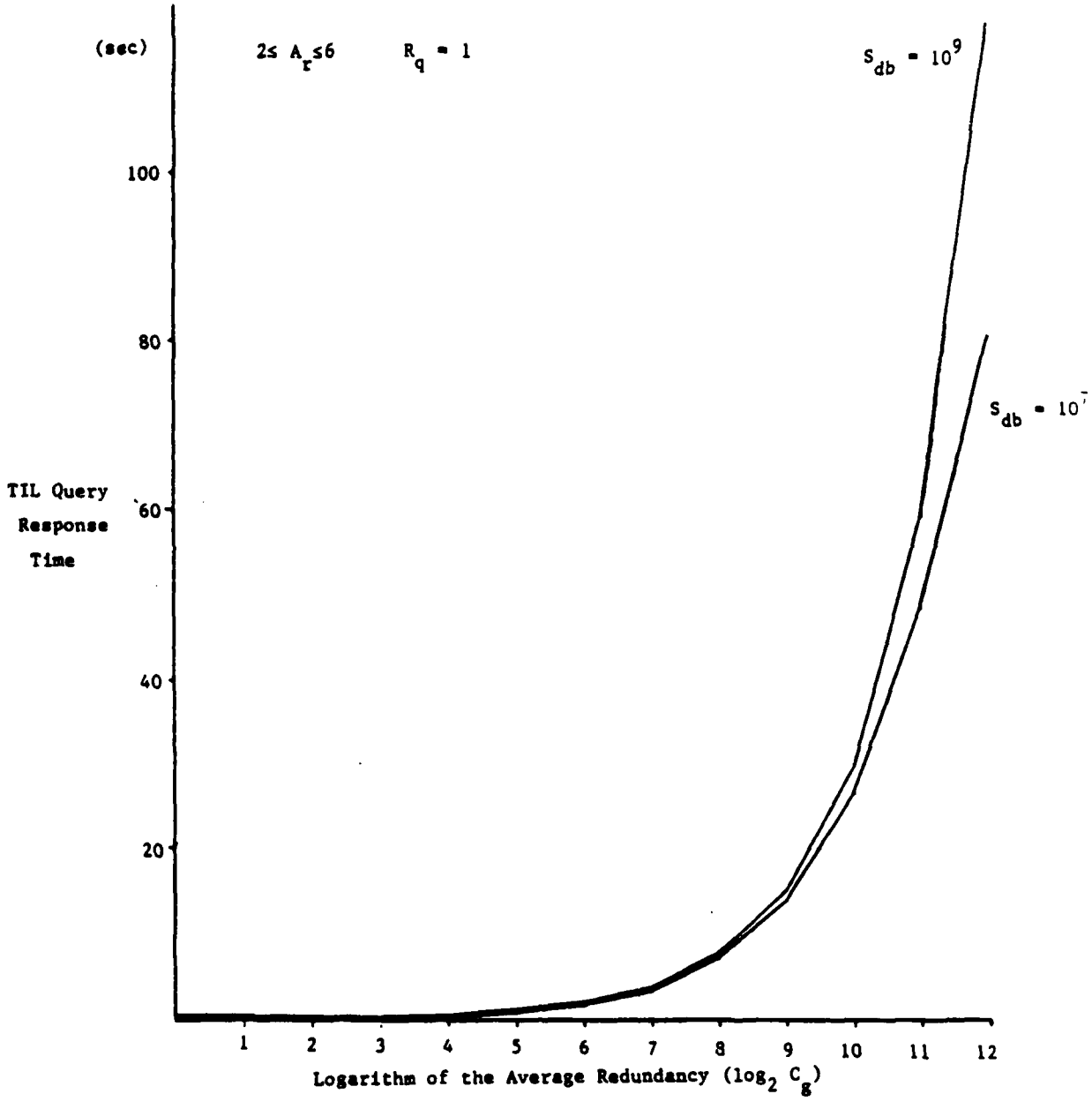


Figure 9.17 TIL1 Query Response Time For Medium and Large DB Sizes (S_{db} =10^7 and 10^9 bytes, R_q = 11)

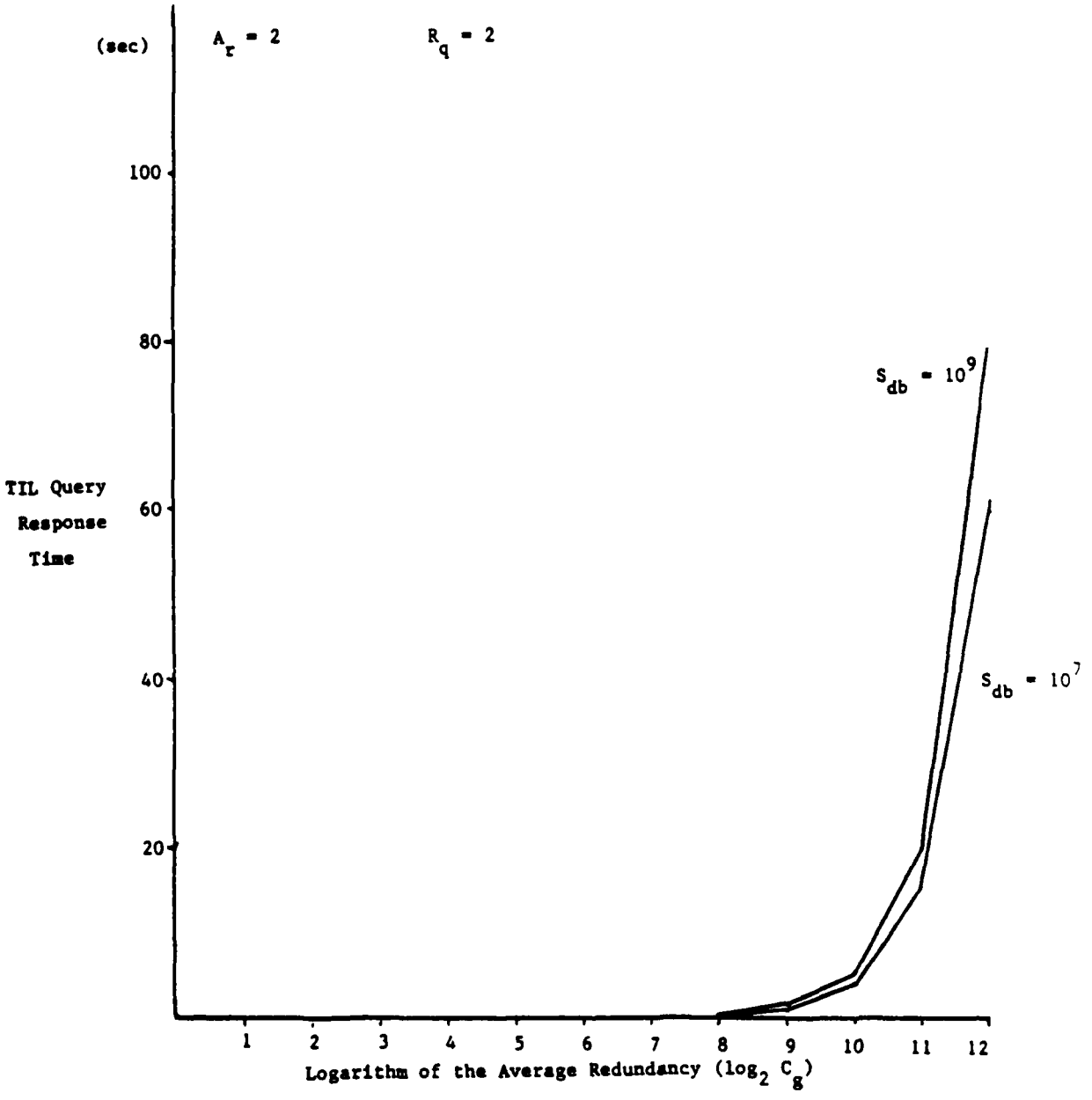


Figure 9.18 TIL1 Query Response Time for Medium and Large DB Sizes (S_{db}= 10^7 and 10^9 bytes, A_r = 2, R_q = 2)

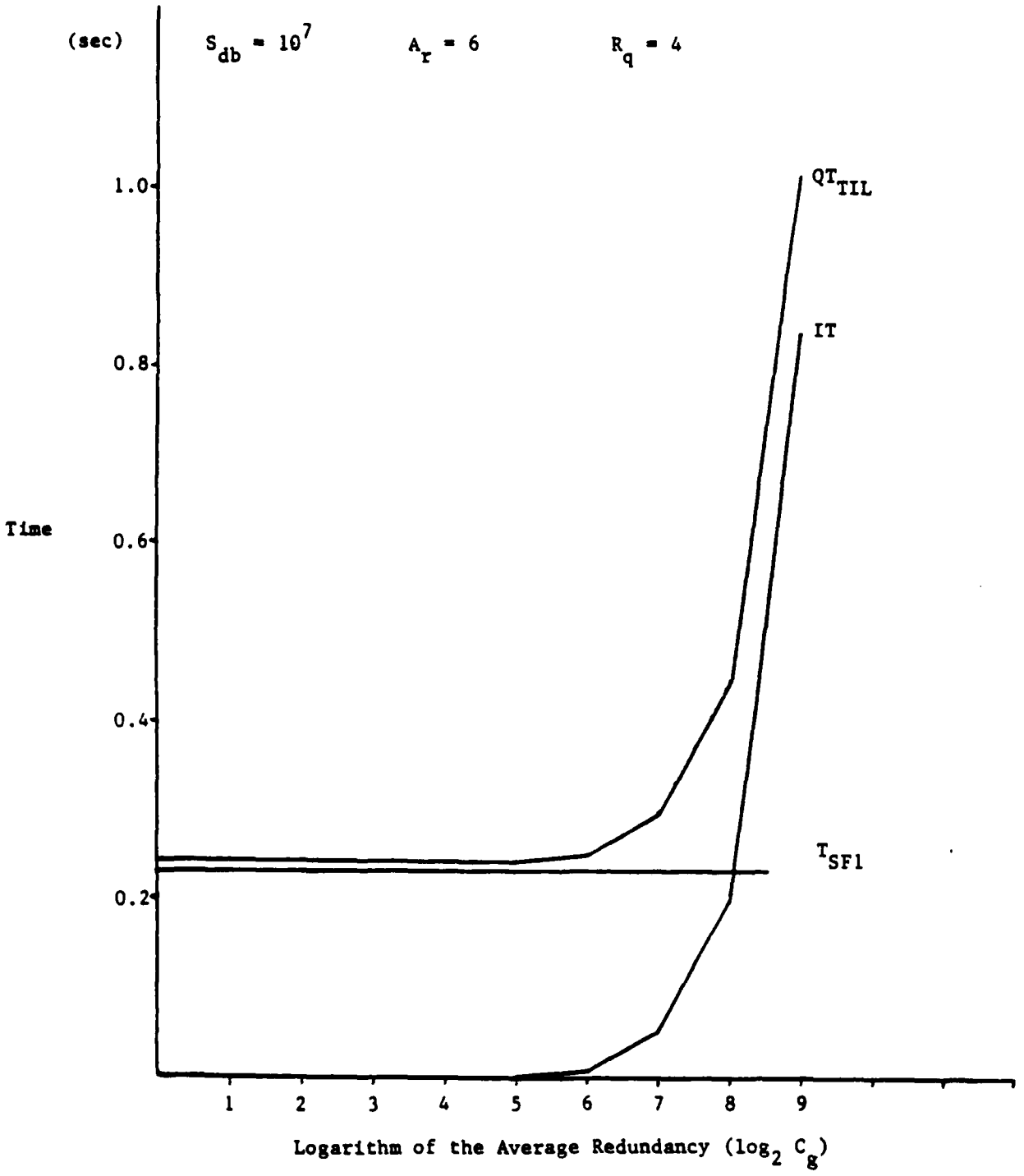


Figure 9.19 Components of the TIL1 Query Response Time ($S_{db} = 10^7$ bytes, $A_r = 6$, $R_q = 4$)

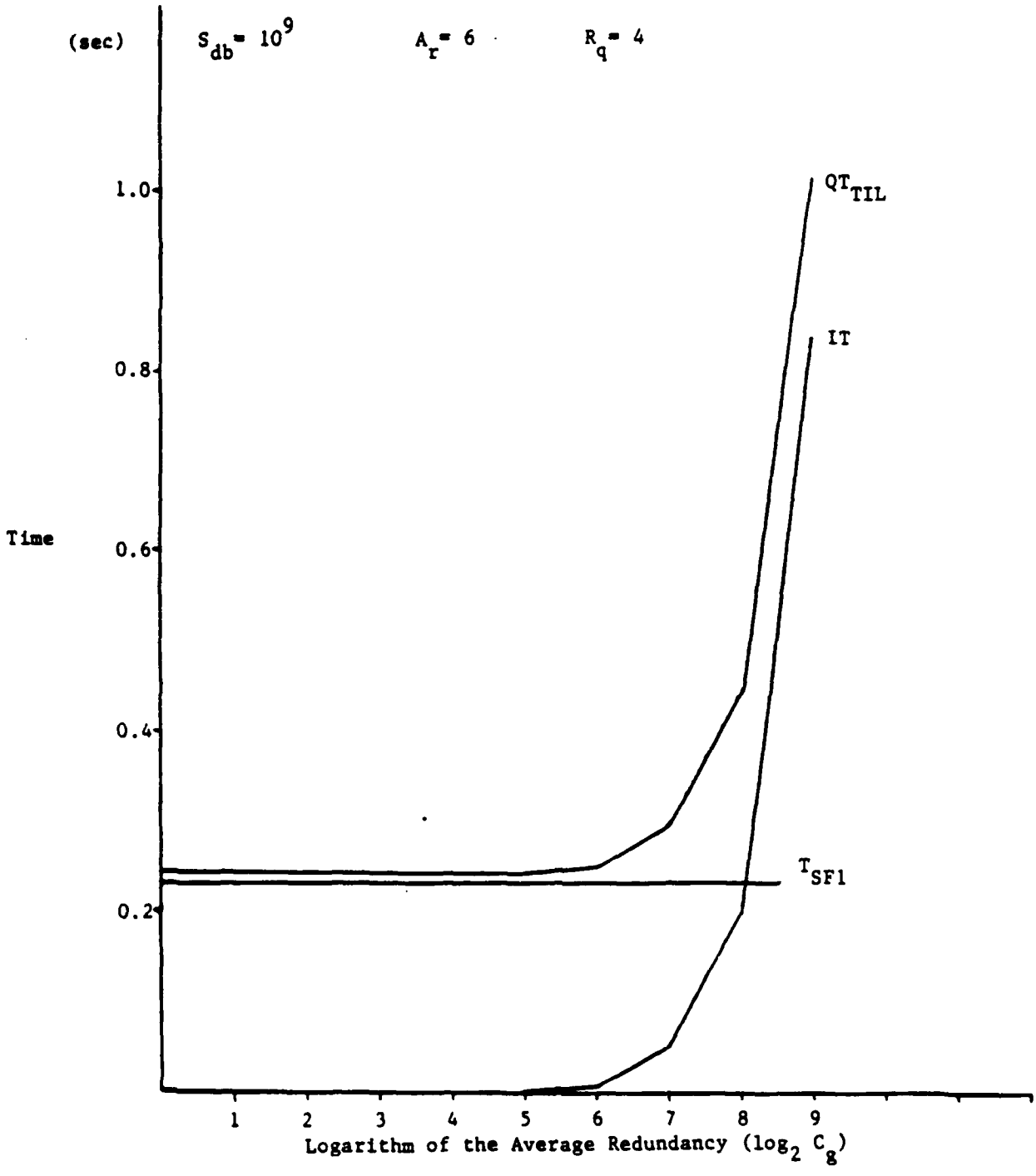


Figure 9.20 Components of the TIL1 Query Response Time ($S_{db} = 10^9$ bytes, $A_r = 6$, $R_q = 4$)

1) For $R_q > 1$, the query response time is shown to increase with C_g at an equivalent rate proportional to C_g^2. Furthermore, the dependency on R_q is not critical as for expected operating points, the intersection time is the prevalent one and tends to converge to a limit that is independent from R_q. Nevertheless, the importance of R_q should not be neglected as it would play a major role when special hardware is designed to handle the TIL1 technique.

2) For $R_q = 1$, the rate at which the TIL1 Query Response Time varies is exponential in nature, mainly due to its dependence on database access time.

3) Finally, if low values for the $\frac{C_g}{N}$ ratio are expected, the total index retrieval time becomes dominant for the TIL1 Query Response Time.

9.2.2. Analysis of TIL2 Time Equations

The intersection and database response time for TIL2 are the same as for TIL1. Globally, TIL2 query response time variations are the same as for TIL1. TIL2 index retrieval time is illustrated in Figures 9.21 to 9.23 for different database sizes, number of arguments in a query and C_g values. The main observations made on TIL2 index retrieval time are:

1) For small and medium database sizes (10^5 and 10^7 bytes), TIL2 requires one additional index file access, per argument in a query, making it slower than TIL1 in the domain of index retrieval time.

2) For larger database sizes (10^9 bytes) and low C_g values, TIL2 index retrieval time is higher than that of TIL1, which is accounted for by the larger surrogate files size. It decreases sharply with increasing C_g due to

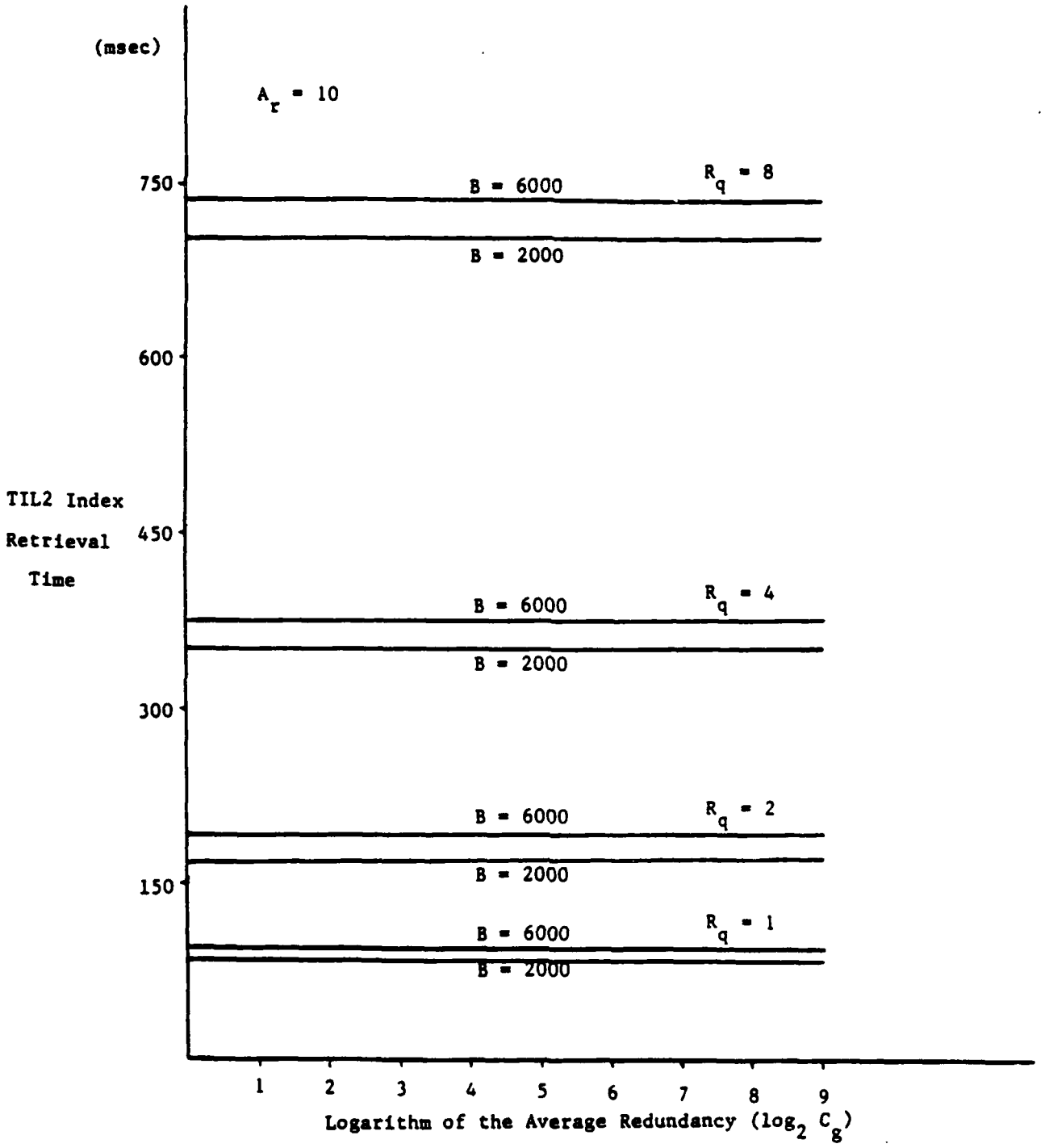


Figure 9.21 Effect of the Number of Arguments in a Query on TIL2 Index Retrieval Time ($S_{db} = 10^5$ bytes)

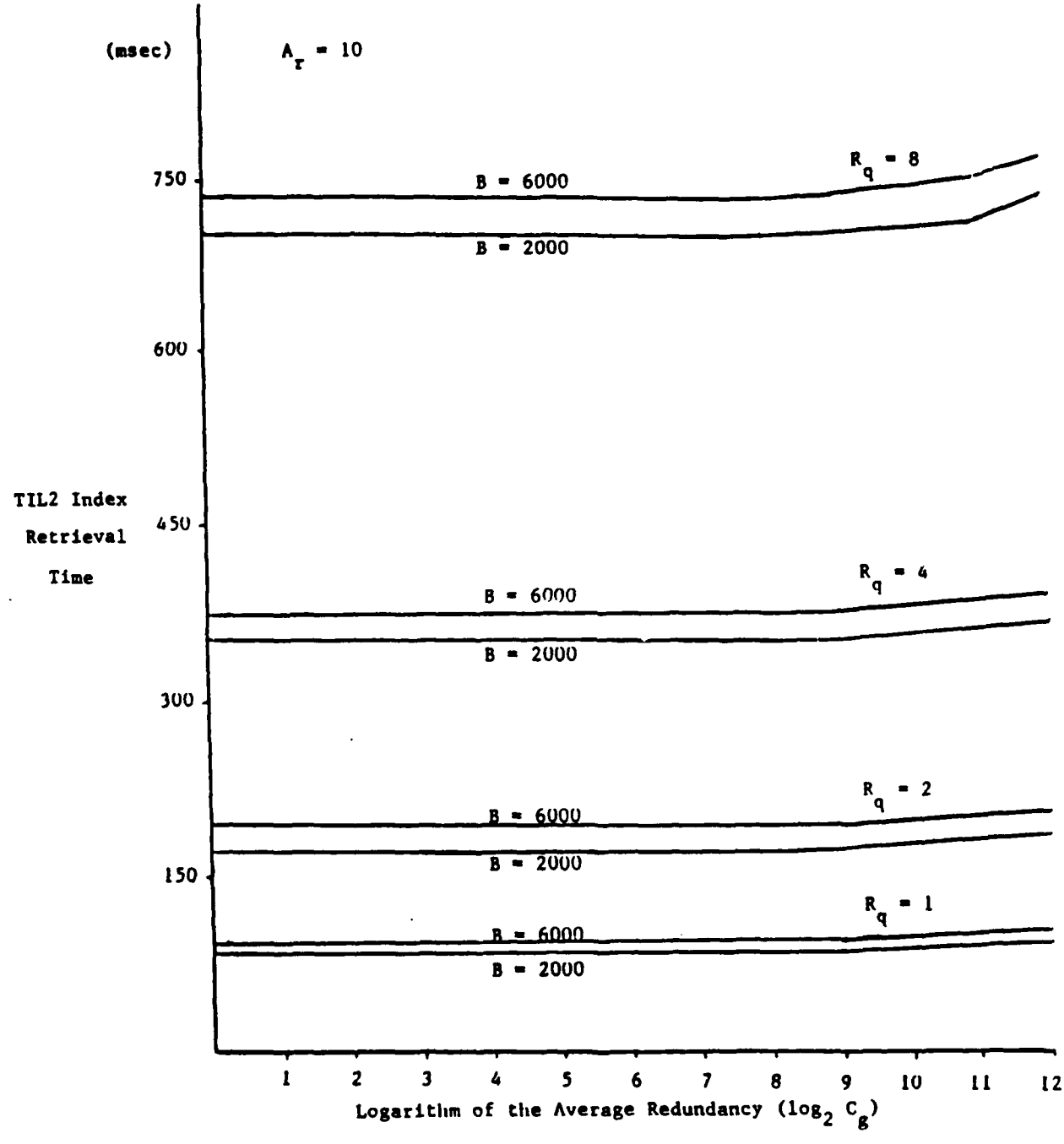


Figure 9.22 Effect of the Number of Arguments in a Query on TIL2 Index Retrieval Time ($S_{db} = 10^7$ bytes)

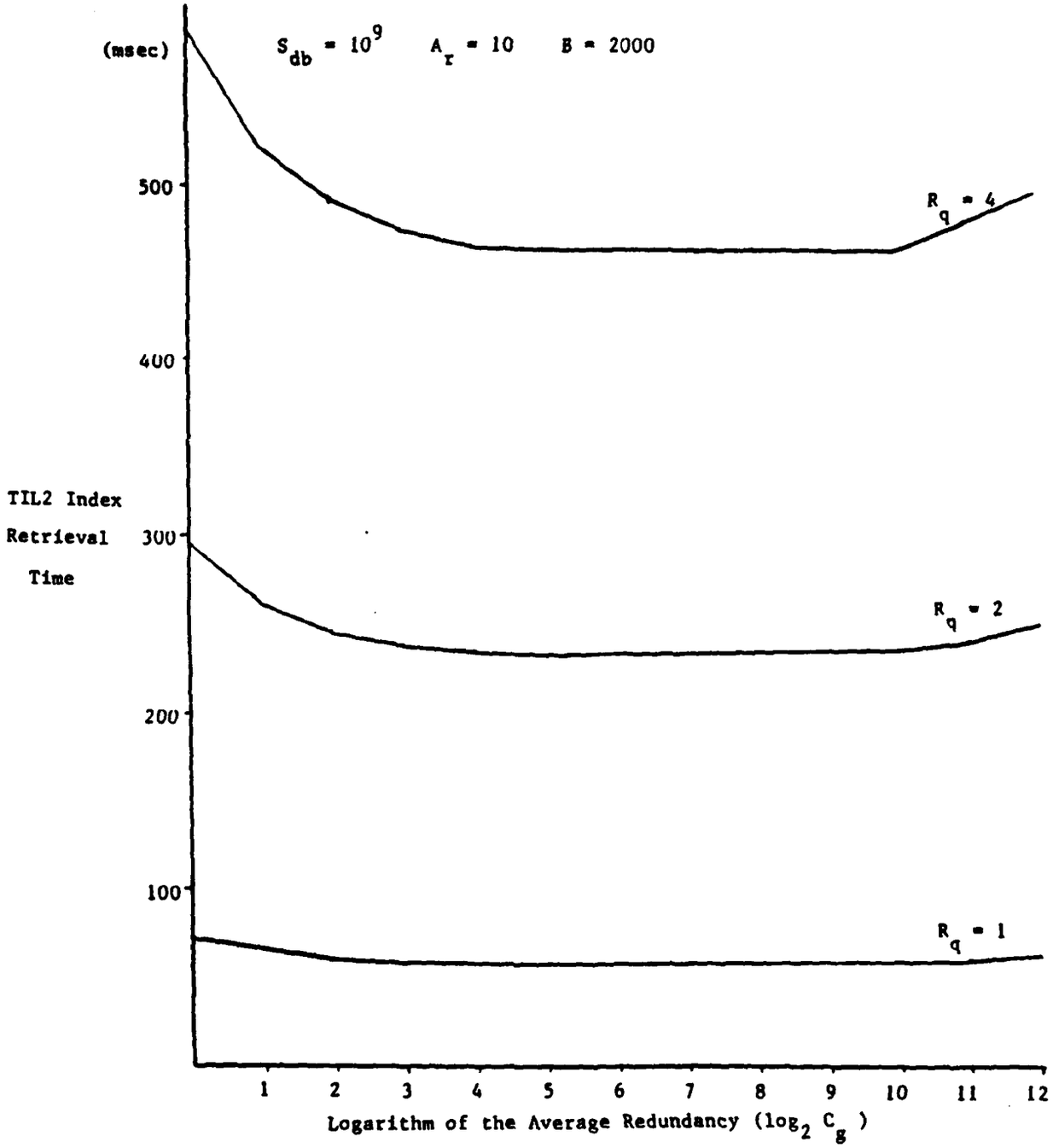


Figure 9.23 Effect of the Number of Arguments in a Query on TIL2 Index Retrieval Time ($S_{db} = 10^9$ bytes)

the fast drop of the surrogate files size. For higher values of C_g, TIL2 and TIL1 index retrieval time are equivalent.

3) Globally, our simulation results showed that, for medium to large values of C_g, TIL2 and TIL1 index retrieval time are equivalent.

The query response time of the TIL2 technique is illustrated in Figures 9.24 to 9.27. Those plots nearly overlap the corresponding ones plotted for TIL1 (Figures 9.14 to 9.20). While slower query response is depicted for low C_g values and small database sizes, it is equivalent to TIL1 technique for larger database sizes, due to the smaller surrogate file size which counterbalances the additional index access per argument in a query.

In general, due the smaller sizes of the surrogate files and similar performance, TIL2 is preferred over TIL1 technique with respect to a size/performance criteria.

9.3. Maintenance Aspects of TIL Surrogate Files

One of the difficulties in using the TIL techniques is their maintenance requirements. Those become a serious drawback, especially in a highly volatile database environment. In such an environment, the vulnerability of the system, as well as the time consuming update process, become major design criteria. Furthermore, the above analysis pertains to a static surrogate file. If, for example, 30% expansion of the main database is forseen, the overall increase of the surrogate files sizes can be greater than 30%, due to the additional increase required for the different record pointers and unique identifiers.

The different maintenance aspects are:

1) Update Operation:

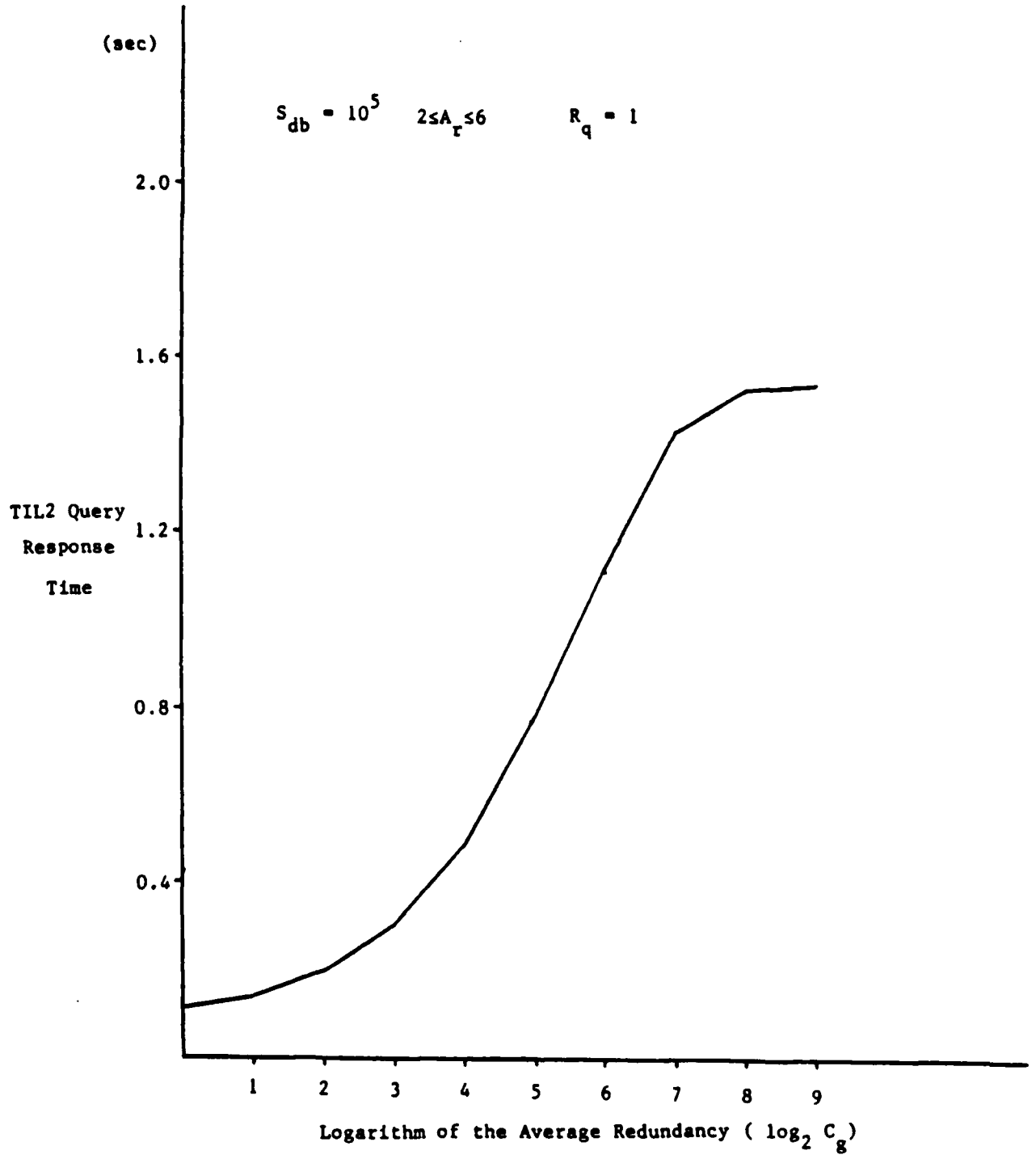


Figure 9.24 TIL2 Query Response Time ($S_{db} = 10^5$ bytes, $R_q = 1$)

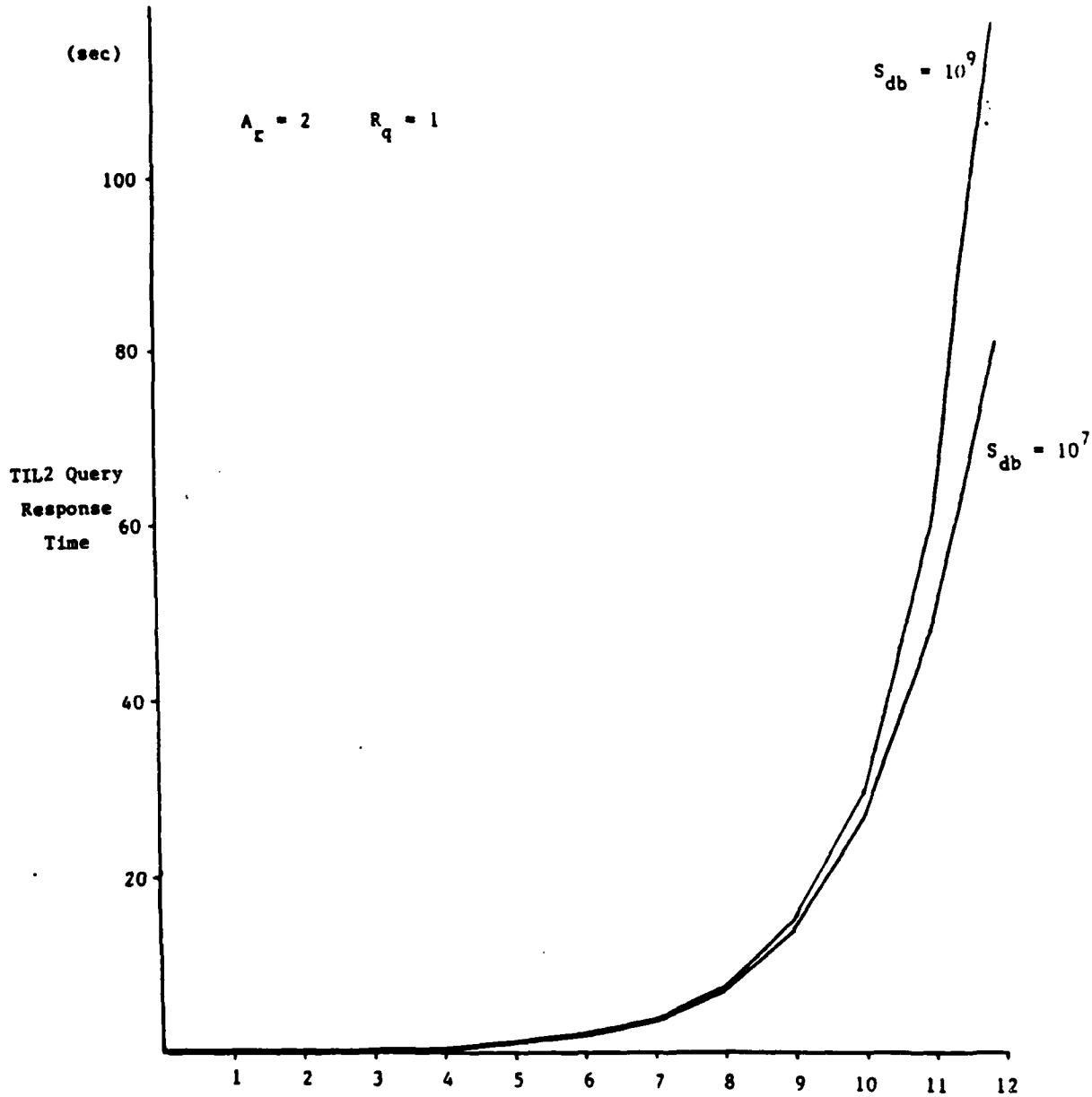


Figure 9.25 TIL2 Query Response Time ($S_{db} = 10^7$ and 10^9 bytes, $R_q = 1$)

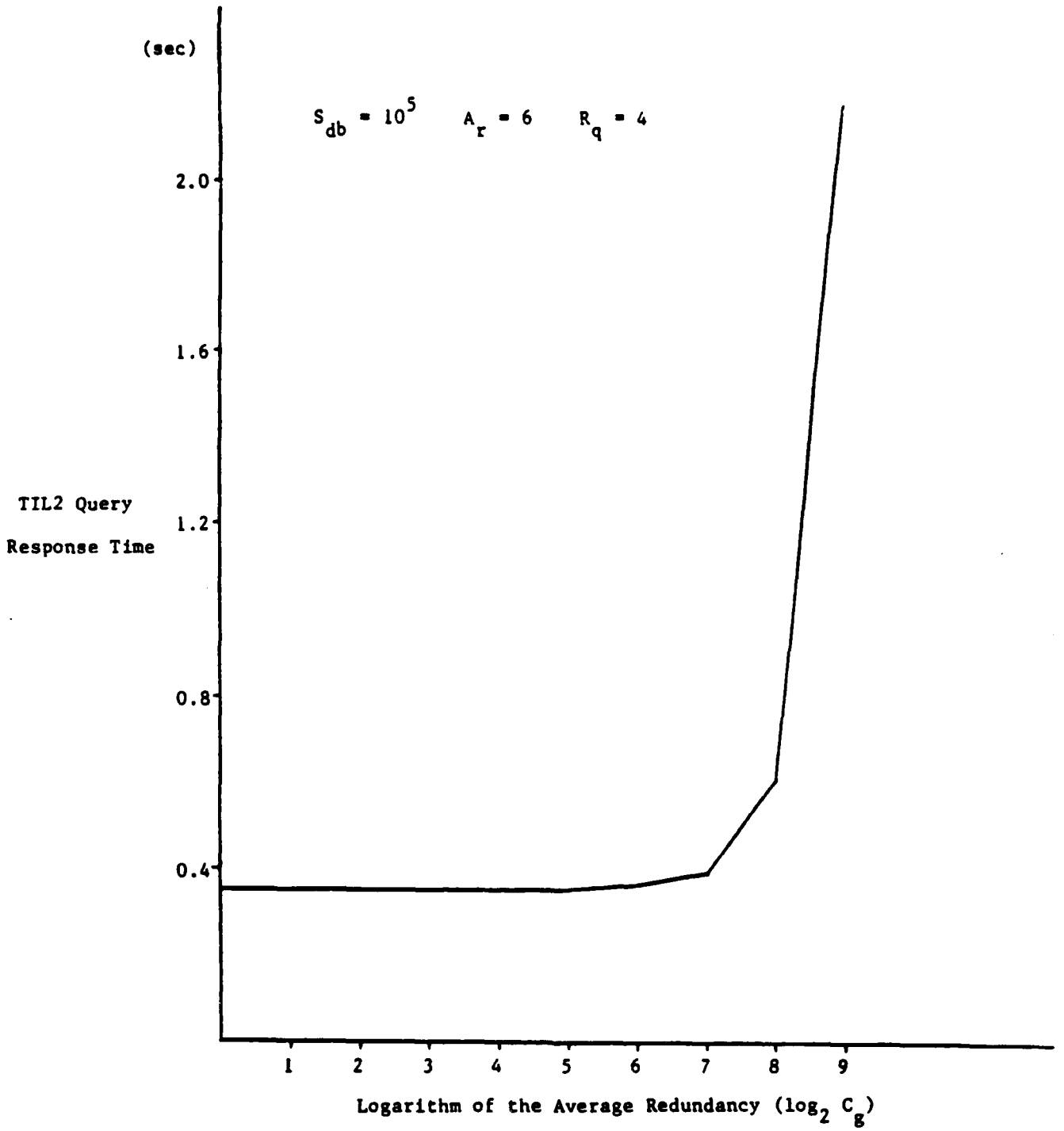


Figure 9.26 TIL2 Query Response Time ($S_{db} = 10^5$ bytes, $A_r = 6$, $R_q = 4$)

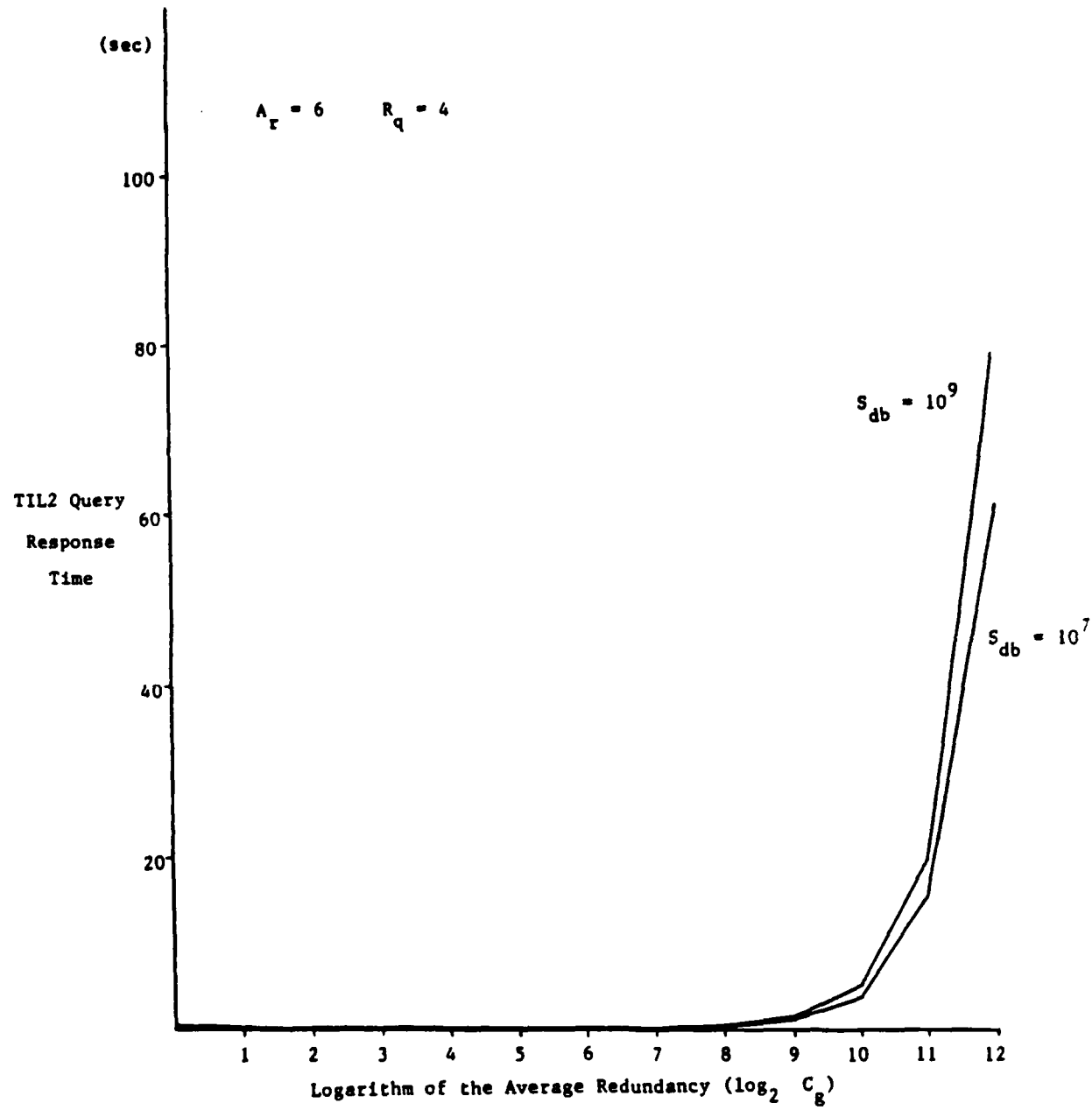


Figure 9.27 TIL2 Query Response Time ($S_{db} = 10^7$ and 10^9 bytes, $A_r = 6$, $R_q = 4$)

When adding a new record to the database, all the index files are to be accessed and reordered; which is a time consuming operation. The use of overflow blocks would decrease the time requirements for update operations with a negative impact on query response time. Block updates could be followed but this technique is not possible in real time databases. In any case, periodical time consuming reordering is necessary. Furthermore, deleting records could be performed by marking techniques and delaying reordering and packing operations to off line maintenance operations periods.

2) Data consistency:

As stated above, access to the whole index files is required in update operations. This tends to increase the system's overall vulnerability and adds to the crash recovery requirements making it a lengthy procedure.

3) Storage requirements:

As analyzed in previous sections, surrogate size requirements in large database environments is high, due to the use of different additional pointer levels for fast main database access.

It can be stated, in general, that the overall management system requirements for TIL Surrogate Files is highly complex and those techniques are not recommended in volatile database environments. Furthermore, TIL1 is better than TIL2 with this respect, due to the fact that it uses two indexing levels rather than the three required for TIL2.

10. Comparison of Surrogate File Techniques

In this section, the different surrogate file techniques are compared as to their respective sizes and query response time. A typical value of A_r=6 and R_q=2 is chosen for the presented graphs for each of database sizes (S_{db}) of 10^5, 10^7 and 10^9 bytes. Furthermore, SCW data is based on an average false drops (DF) of one and R_q=2.

10.1. Size Comparison of SCW, CCW, TIL1 and TIL2

From Figures 10.1 to 10.3, one concludes that CCW and SCW dominate TIL techniques as to the required storage of the surrogate files. Furthermore, it is noted that CCW surrogate files are smaller than those for SCW for medium to high average redundancy factor (C_g). Realistic values of 10 to 20 % of S_{db} for those techniques are observed, while the requirements for TIL techniques are in the 15 to 30 % range as observed from Figures 10.2 and 10.3.

Furthermore, it is observed that larger TIL surrogate file sizes are needed in actual systems to allow proper management while surrogate files for SCW and CCW techniques expand by a simple addition to the end of the file.

10.2. Query Response Time Comparison of SCW, CCW, TIL1 and TIL2

Different observations are made on Figures 10.4 to 10.6 pertaining to the query response time of the different surrogate files techniques.

1) For small database sizes as in Figure 10.4, SCW and CCW dominate TIL techniques as to the query response time. This is due to the small surrogate file sizes under consideration that do not necessitate the complex TIL techniques' access scheme. In this case, sequential access and search of the

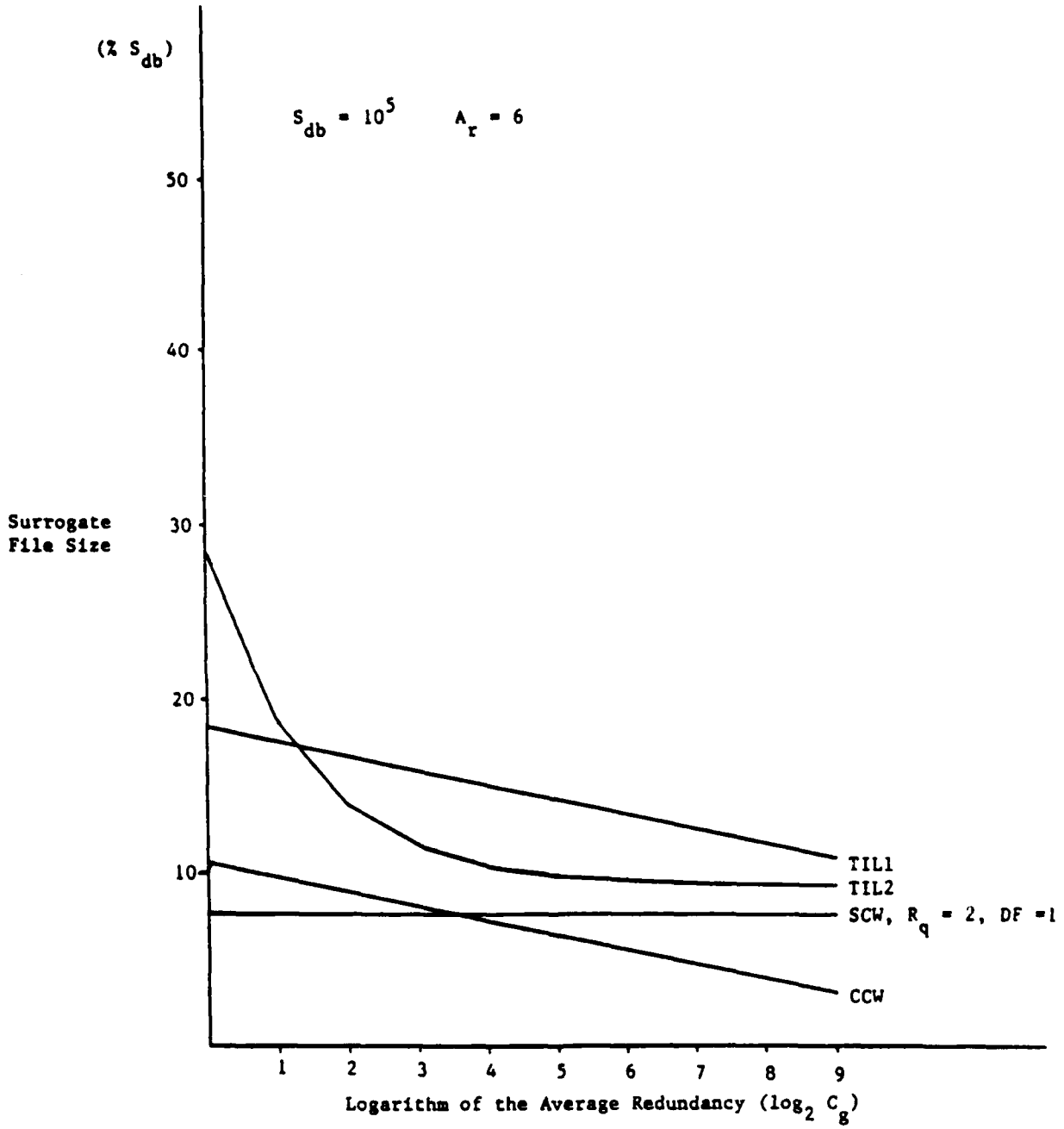


Figure 10.1 Surrogate File Size Comparison ($S_{db}=10^5$ bytes)

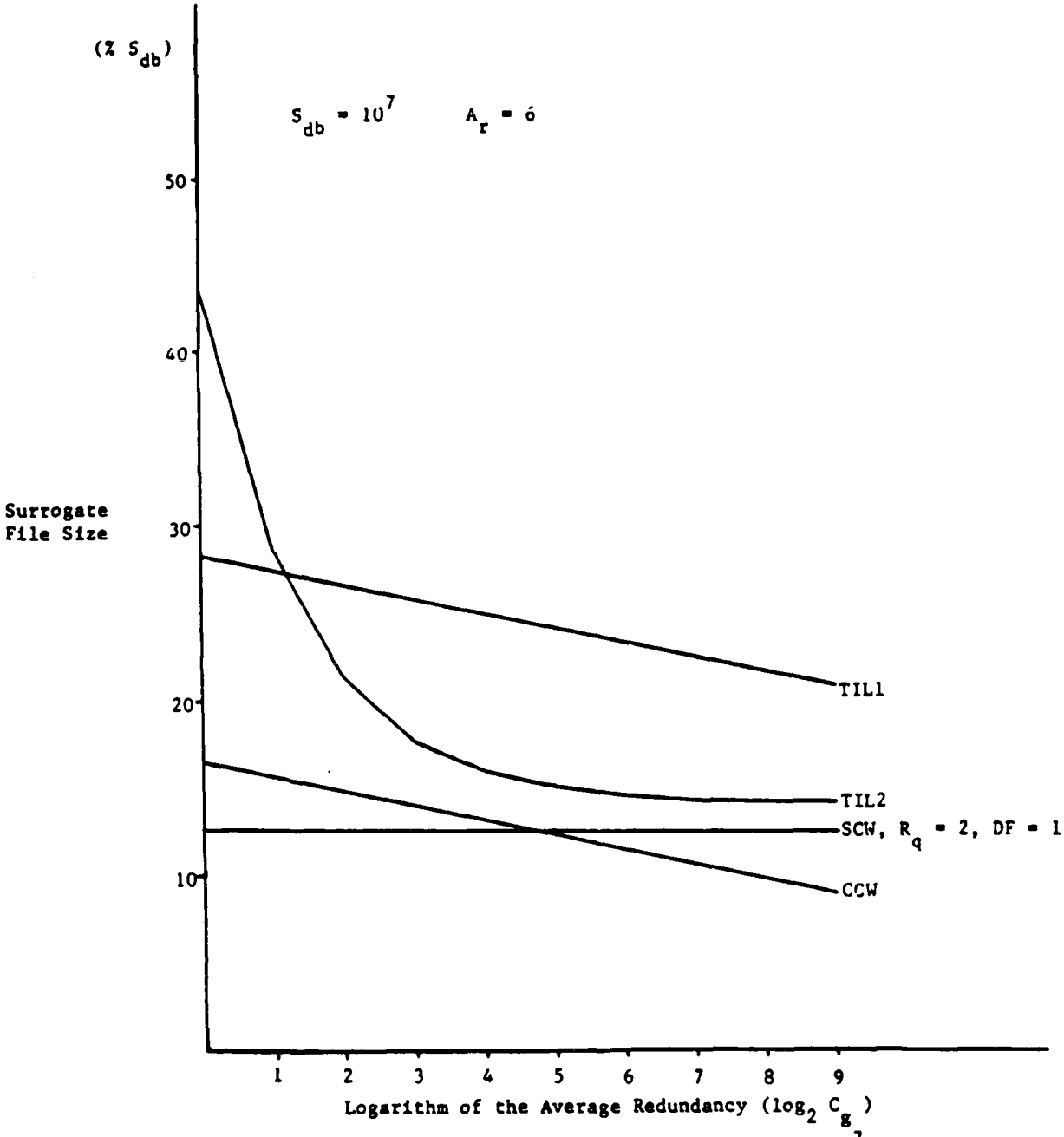


Figure 10.2 Surrogate File Size Comparison ($S_{db} = 10^7$ bytes)

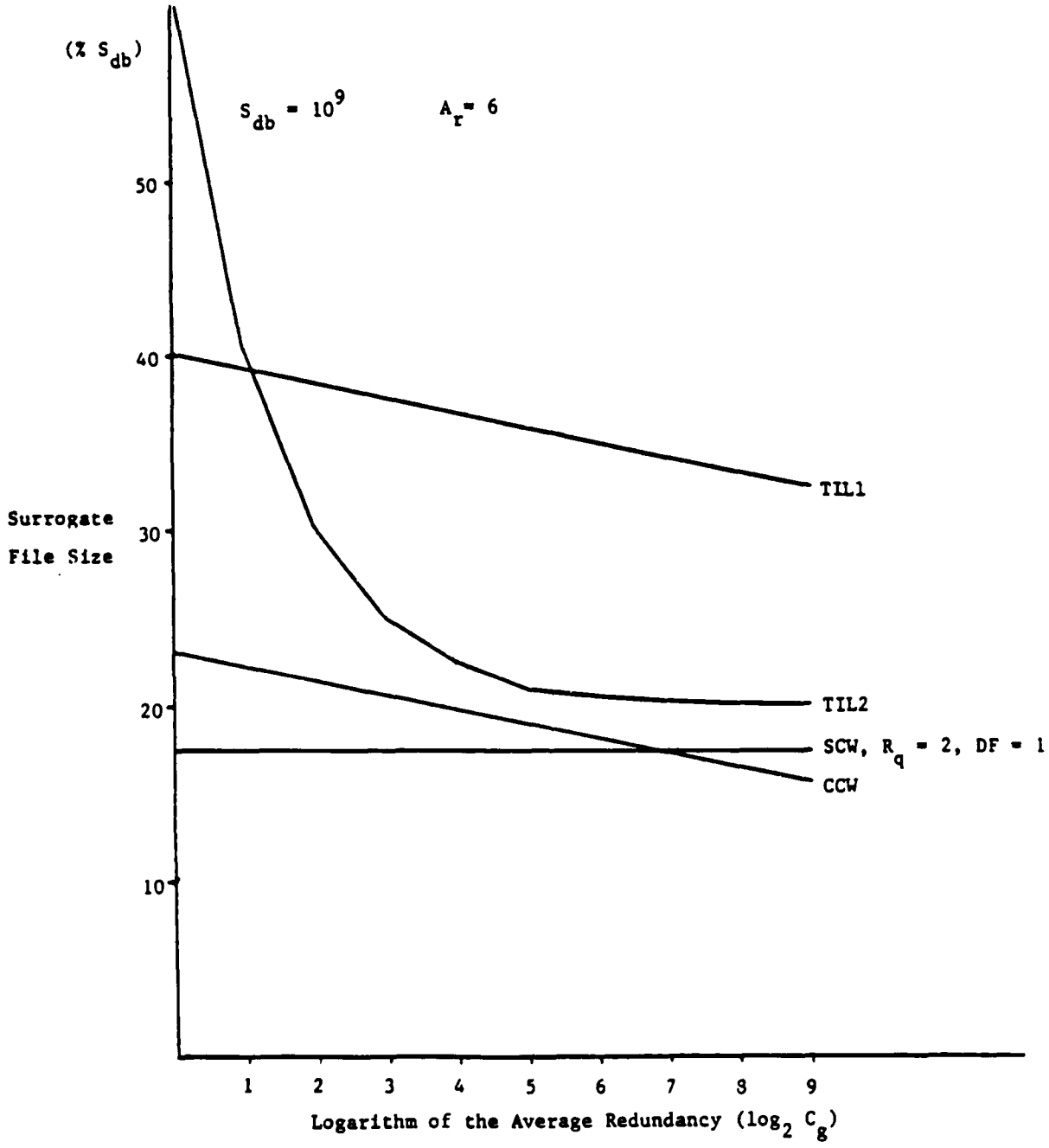


Figure 10.3 Surrogate File Size Comparison ($S_{db} = 10^9$ bytes)

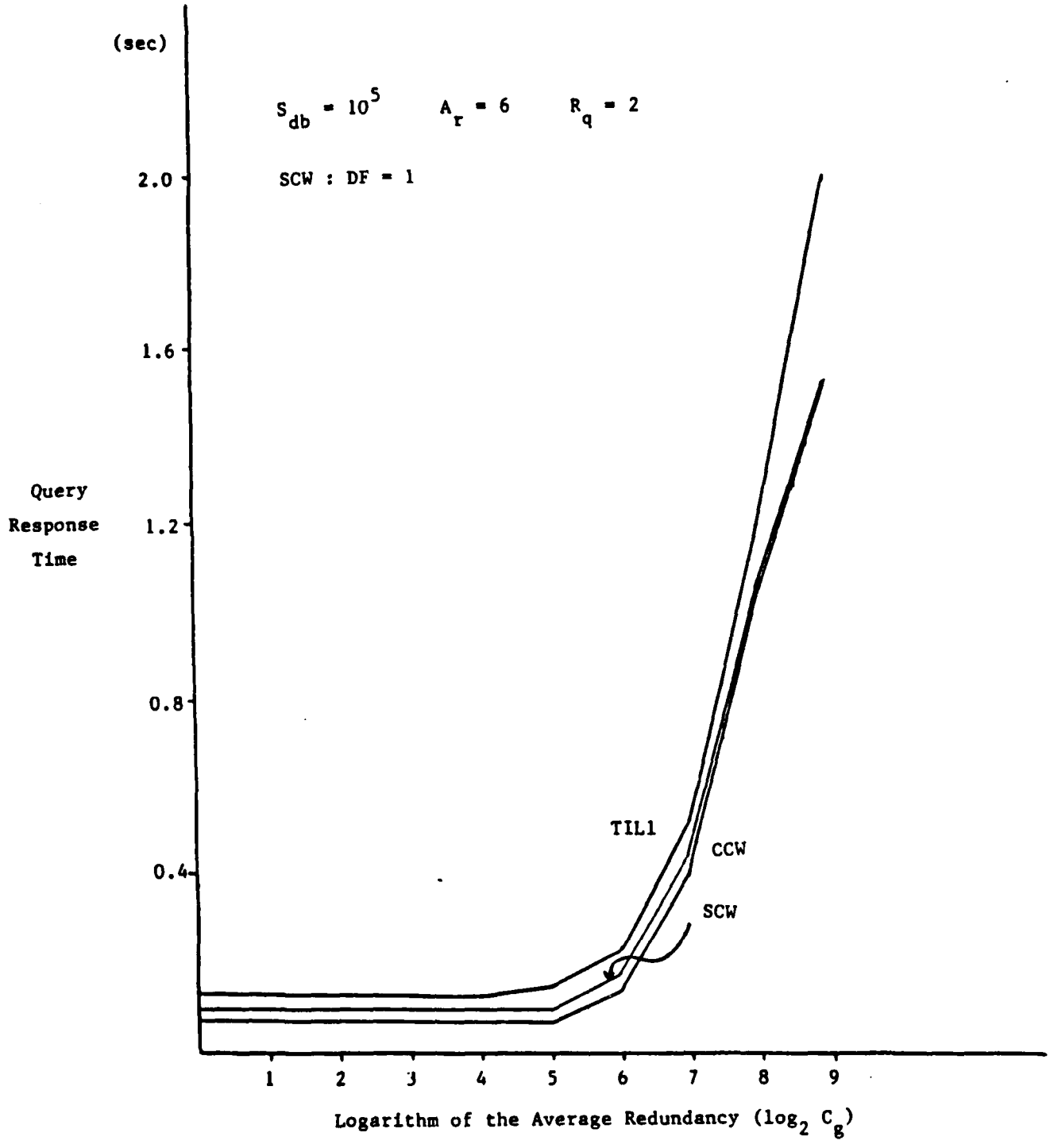


Figure 10.4 Query Response Time Comparison ($S_{db} = 10^5$ bytes)

surrogate files, as performed with SCW and CCW is faster than the index access scheme of transformed inverted lists.

2) For medium database sizes, TIL techniques are efficient as to query response in the medium to low range of the average redundancy factor C_g. This is reflected in Figure 10.5 for $S_{db}=10^7$ and $C_g \leq 9$ where it is observed that TIL techniques dominate SCW and CCW techniques. For larger C_g values, the situation reverses and TIL techniques are dominated by CCW and SCW techniques. This behavior is due to the high intersection time required for large values of C_g. The cut point in Figure 10.5 being around $\log_2 C_g=9.5$.

3) Figure 10.6 is typical of large database sizes. In this case it is noted that TIL techniques are more appropriate than CCW and SCW techniques. For low values of C_g ($\log_2 C_g<9$). the speed up of TIL versus SCW and CCW techniques is greater than 300, while this value drops very fast to the 2.5 to 4 range for $\log_2 C_g=12$. This is due to the fast change in required intersection time for the TIL techniques. It can be easily extrapolated that there exists two cut points for the TIL query response time curve with that of CCW and SCW curves respectively around $\log_2 C_g=13$ whereby CCW and SCW techniques would become faster than transformed inverted lists.

It is worth noting that the negative slope of the CCW query response time curve in Figure 10.6 is mainly due to a smaller number of bits for the binary representation (BR) of each argument value as C_g increases. The average value C_g should be chosen accurately at the design level of the CCW Surrogate Files as it affects the expected response time for large database sizes.

In the next Section, we discuss possible computer architectures for the processing of the surrogate files and in Section 12 briefly analyze the possible implementation of relational operations on the surrogate files.

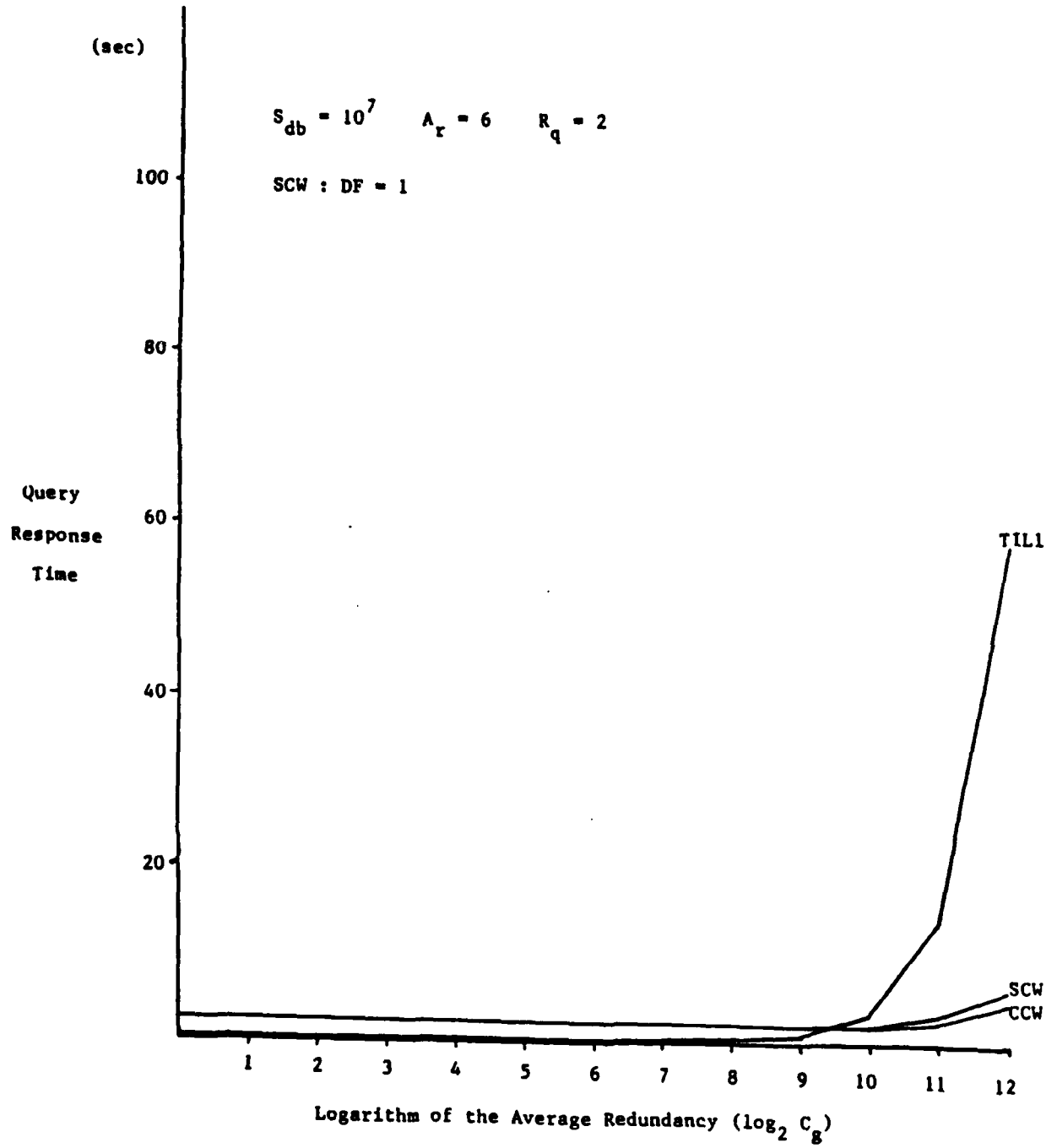


Figure 10.5 Query Response Time Comparison ($S_{db} = 10^7$ bytes)

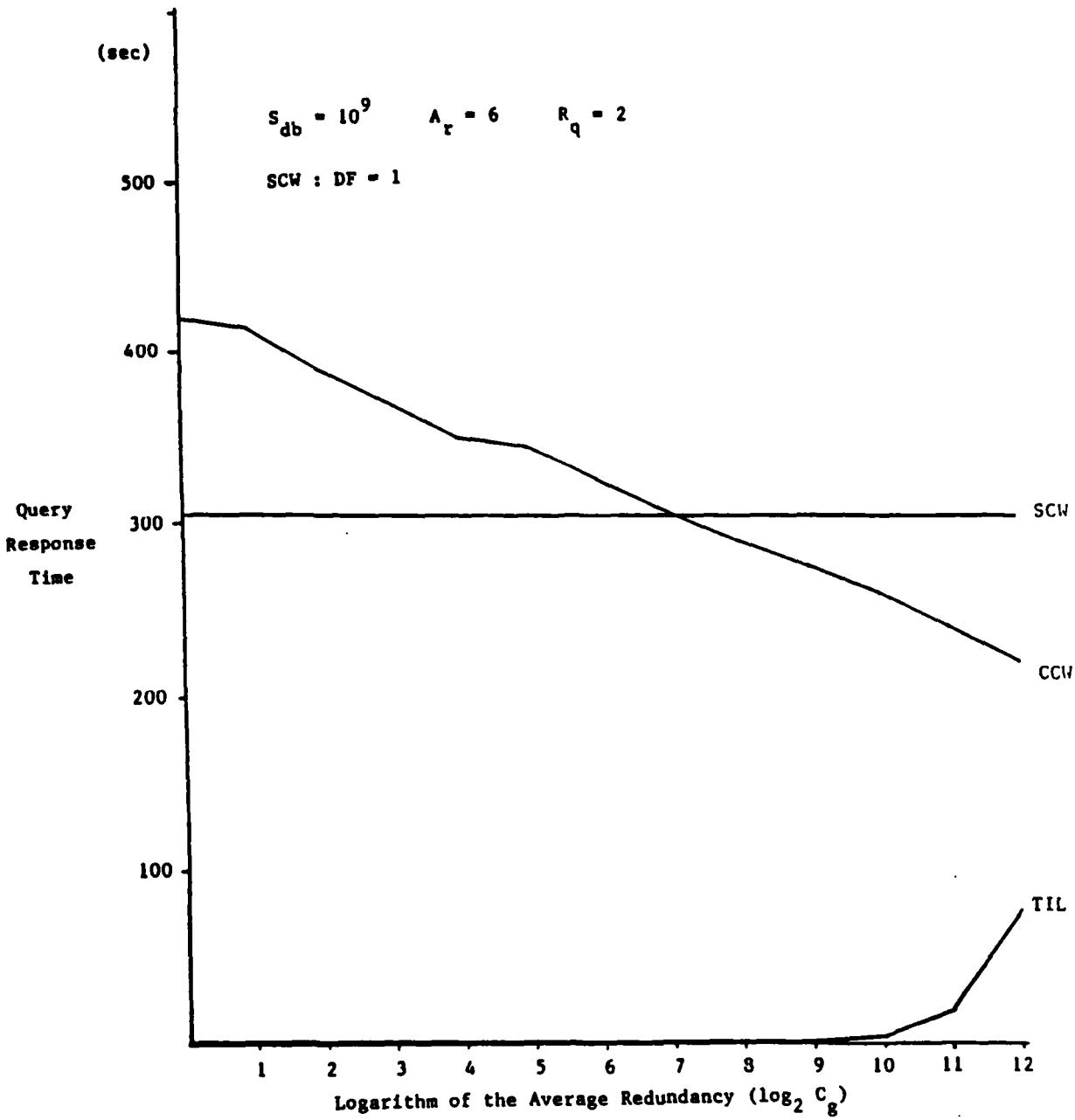


Figure 10.6 Query Response Time Comparison ($S_{db} = 10^9$ bytes)

11. Computer Architecture for Surrogate File Processing

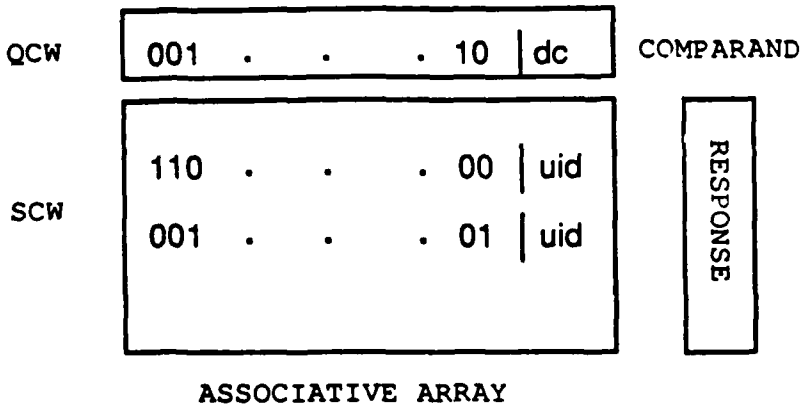
11.1. Associative Memory

In this section we will consider the processing of the surrogate file by an associative memory when the file is constructed using SCW, CCW or TIL. Recall that under our previous assumptions all partial match retrievals will be performed using the surrogate file. Thus, retrieval of the facts will take at most two disk accesses given the unique identifier.

Associative memories have the capability of performing a wide variety of operations on array resident data. These operations include exact match, maximum, minimum, etc. as well as Boolean operations. These devices are well known for performing these operations using content addressing and parallelism. However, these devices are generally costly, rigid in data format and input/output bound (Berra 1979). Nevertheless, when the associative memory is used to process the surrogate file, one of these disadvantages goes away and another is mitigated. That is, the format of the surrogate file is highly regular and maps very well into the associative memory. Also, the surrogate file is a compact representation of the data being considered and less I/O is required.

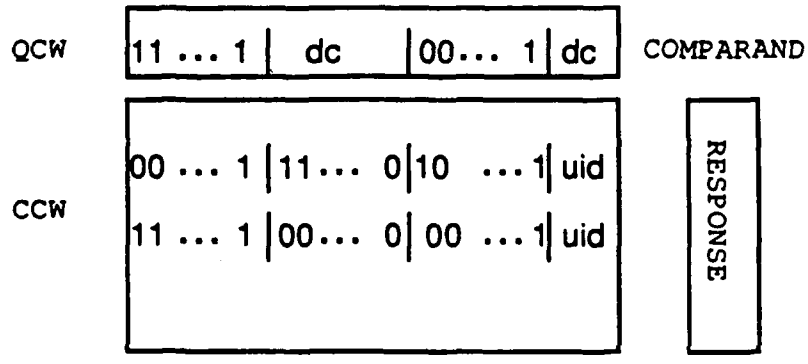
In the case of SCW, research by Ahuja and Roberts (1980) indicates that the mapping between an SCW file and the associative memory is very good. The processing of the array resident data takes place by performing an exact match search using the QCW as the search argument as shown in Figure 11.1.a. The unique identifiers for all of the SCW's that matched the QCW on the basis of ones would be sent to the fact manager for retrieval and further processing of the facts.

The processing of a CCW file would be similar but with some variation as



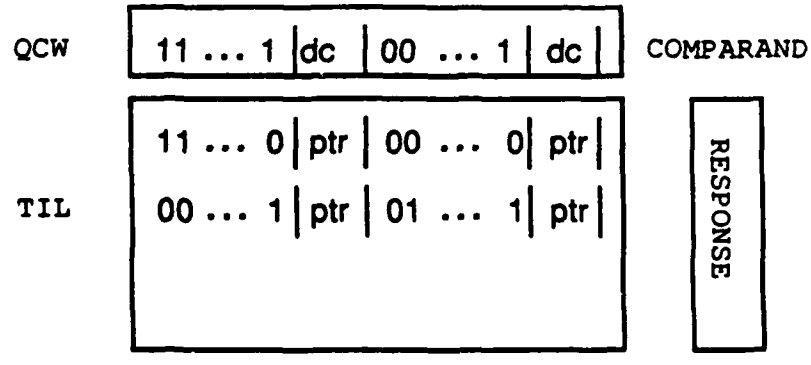
Search on the basis of ones

Figure 11.1.a SCW Mapped into Associative Memory



Search on the basis of ones and zeros

Figure 11.1.b CCW Mapped into Associative Memory



Search on the basis of ones and zeros

Figure 11.1.c TIL Mapped into Associative Memory

shown in Figure 11.1b. The QCW would have don't cares in those positions where values are unknown and a known bit pattern in the positions of interest. An exact match search would be performed on the ones and zeros of the fields of interest, ANDed together and the selected unique identifiers sent on further processing.

In order to process TIL files using an associative memory, as shown in Figure 11.1.c, one would need to form subfiles of unique identifiers for each value in each argument position since these values will not be unique. That is, each distinct value will have a pointer attached to it that points to the list of unique identifiers of the facts containing that value. The processing of the TIL file will require less transfer of data to the associative memory for searches with few arguments, but will require intersecting the lists of unique identifiers prior to passing the results on. However, if many arguments are given in the request then considerable processing will be required.

11.2. Back End System

Shown in Figure 11.2 is a back end system for the management of a very large extensional database of facts. The back end system will also manage many intentional databases but those are not shown on the diagram. We assume that there are many gigabytes of fact data stored on the EDB disks. Likewise, there are several gigabytes of surrogate file data stored on the SF disks. Since we have assumed the relational model we will store the facts by relation and then by tuple unique identifier within relations. As previously mentioned we will access only by relation name and then by tuple identifier, so extendible hashing or some such technique that minimizes disk accesses can be used. The surrogate file will really consist of many subfiles one for each relation. Each of the subfiles will be distributed uniformly over all of the SF disks in order to insure maximum parallelism

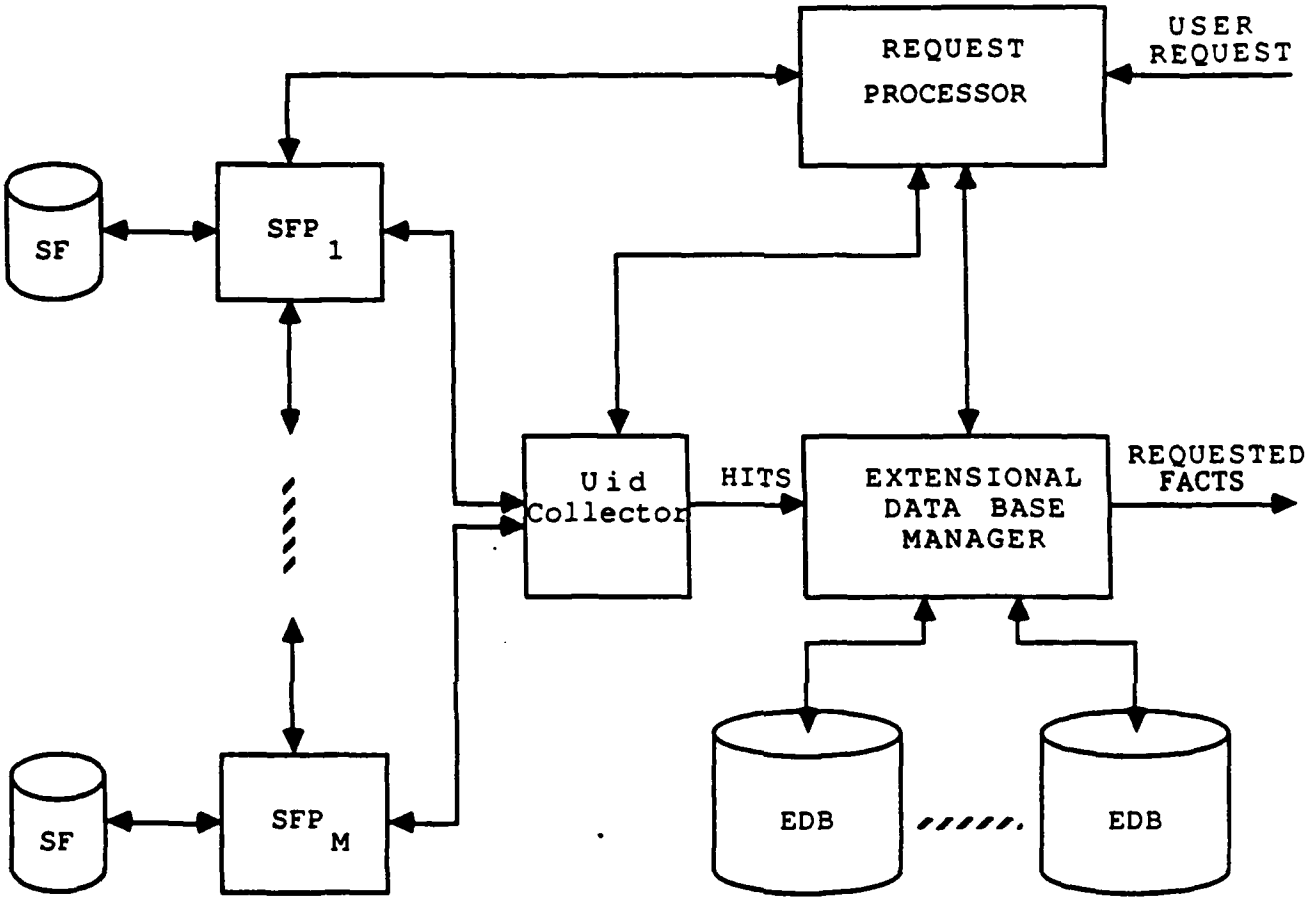


Figure 11.2 Back End System for Fact Management

in disk accessing. As an example, suppose that the surrogate file is constructed with concatenated code words and the user is interested in retrieving fact data given some subset of values from a particular relation. The query code word would be constructed in the Request Processor using the proper hashing function and considering the positions of the values within the relation. The QCW would then be broadcast to all of the Surrogate File Processors (SFP) to be used as a search argument. As discussed in the previous section, one could think of the SFP as an associative memory with the QCW as the search argument. However, the full capabilities of the associative memory would not be needed for the operations discused here. There is no point in providing much more capability in the SFP than is required to process the surrogate file data at disk rates. Thus, the SFP would be rather simple since all it needs to do is compare the QCW with each CCW at disk rates and strip off the unique identifiers. As soon as any unique identifiers are found by the SFP's they can be sent to the collector and quickly passed on to the Extensional Data Base Manager (EDBM) for processing, thus providing some level of pipelining. The EDBM will retrieve the facts, compare them with the search criteria to insure that a collision has not occurred, put them in blocks, and send the blocks to the logic programming engine.

One would follow a similar process if the surrogate file was constructed with SCW. The main difference is that extra processing would be required by the EDBM because of the false drops. Other work regarding processors for SCW's can be found in Lee (1986) and Colomb and Jagasooriah (1986).

If the surrogate file is composed of TIL's the processing will be slightly different and an additional hardware unit will have to be added to the system shown in Figure 11.2. Each of the lists would be distributed uniformly over the SF disks. As an example assume that a user's request requires access to only two

lists. The first list would be input to the SFP's and a simple comparison is made for an exact match. Note that the SFP is just a simple comparator in this case. The unique identifiers would be stripped off and sent to the collector. The second list would be processed in a similar way. However, now the two lists will have to be intersected. Thus, we will need an additional block between the collector and the EDBM called the Intersector.

In this discussion we have tried to provide an idea of how such a system might work in retrieving facts from a very large extensional database. Obviously, many more issues need to be addressed such as updating, integrity and collisions.

12. Relational Operations on the Surrogate File

A serendipitous benefit obtained from using the surrogate file approach is that some relational operations can be performed on the surrogate file rather than on the relations themselves. This offers considerable potential savings in time to carry out these relational operations and thus contribute an additional measure of performance to the system.

Consider the case in which the surrogate file is constructed with CCW's. To perform a selection on a particular field, one just uses the surrogate file processor to do a partial match on just that field. Extending this slightly, it is clear that the SFP is just a selection mechanism. With a small amount of extra hardware, the SFP could be called the selector.

Consider the performance of a join using the hash join algorithm. Since the surrogate file already consists of hashed values, all that needs to be done is to assign the portion of the code word that represents the join variable and the unique identifier for each relation to buckets. Then, based on matching within each bucket, which can be done in parallel, pairs of unique identifiers can be sent to the EDBP for final verification. This procedure would work for the equijoin but an order preserving hashing function (Garg and Gotlieb 1986) would be needed for other joins.

It appears that there would be no advantage to doing projections on the surrogate file. Thus, the EDBP would have to have tuples stored in such a way that projections could be done directly on the tuples of the relation under consideration. This could be accomplished by treating each relation independently with its own extendible hashing system. Storing the relation this way would also aid in the performance of other operations that pertain to all or a large number of

tuples in the relation.

Finally, range queries could be performed on the CCW file given a suitable order preserving hashing function; albeit with some additional processing such as sorting.

In the case of SCW it would be difficult to perform most relational operations on the surrogate file. In the case of TIL files selections and hash joins could be performed but projections would have to be performed on the tuples themselves. Range queries could also be performed given a suitable order preserving hashing function and some additional processing.

13. Conclusion

Three techniques for structuring surrogate files are compared in terms of storage requirement, retrieval performance and maintenance. For large Extensional Database files, the techniques using TIL files is superior to the others in retrieval performance. However, the size of TIL files are larger than those of others and the maintenance of TIL files causes the large overhead. Also, CCW and SCW techniques can easily be implemented with associative memory and parallel processors to improve the retrieval performance.

The size and retrieval time of the CCW is smaller than those of the SCW technique when the average number of arguments in a query is small. Additionally, most of the relational operations can be performed on the CCW file itself and the CCW technique does not produce false drops while the SCW technique produces false drops inherently.

A back end system for the management of large Extensional Databases is introduced. Our future research shall be towards the development of the proposed back end system, based on the current results. A design around CCW and SCW techniques shall stress solutions for the speedup of the retrieval time for large databases, while the TIL based system shall stress solutions to the intersection time bottleneck and the complex management requirement for TIL techniques.

Appendix 1: Average Number of Adjacent Blocks Containing the Same Argument Value

The average number of consecutive blocks containing records with the same i_th argument value in an index file is computed in this section.

The following terms are defined:

1) C_i: the number of records with the same i_th argument value.

2) C_b: the number of records in an index file block.

3) X: number of consecutive blocks as a random variable. We have to compute E(X), the expected value of X.

It is noted that the C_i records reside consecutively in an index file, and the first record can be located at any k_th position in an index block with equal probability.

Three cases are considered:

1) $C_i < C_b$:

The number of blocks to be retrieved is either 1 or 2 blocks. We can write:

$$P(X = 2) = \frac{C_i - 1}{C_b}$$

$$P(X = 1) = 1 - \frac{C_i - 1}{C_b}$$

Then the expected value of X is:

$$E(X) = 2 \times \frac{C_i - 1}{C_b} + 1 \times (1 - \frac{C_i - 1}{C_b})$$

2) $C_i > C_b$, with $C_i = n \times C_b + r$ and $r \neq 0$:

We can write:

$$P(X = \left\lceil \frac{C_i}{C_b} \right\rceil + 1) = \frac{(C_i \mathrm{mod} C_b) - 1}{C_b}$$

$$P(X = \left\lfloor \frac{C_i}{C_b} \right\rfloor) = 1 - \frac{(C_i \mathrm{mod} C_b) - 1}{C_b}$$

The expected value of X (i.e average number of blocks) is:

$$E(X) = (\left\lceil \frac{C_i}{C_b} \right\rceil + 1) \times \frac{(C_i \mathrm{mod} C_b) - 1}{C_b} + \left\lceil \frac{C_i}{C_b} \right\rceil \times (1 - \frac{(C_i \mathrm{mod} C_b) - 1}{C_b})$$

where $C_i \mathrm{mod} C_b = C_i - \left\lfloor \frac{C_i}{C_b} \right\rfloor \times C_b$.

3) $C_i = n \times C_b$:

In this case we write:

$$P(X = \frac{C_i}{C_b} + 1) = \frac{C_b - 1}{C_b}$$

$$P(X = \frac{C_i}{C_b}) = \frac{1}{C_b}$$

The average number of blocks is then:

$$E(X) = (\frac{C_i}{C_b} + 1) \times \frac{C_b - 1}{C_b} + \frac{C_i}{C_b} \times \frac{1}{C_b}$$

Case 1 and 2 are governed by the same formula derived in case 2. The effect of case 3 can be combined in the following general formula:

$$E(X) = (\left\lceil \frac{C_i}{C_b} \right\rceil + 1) \times \frac{(C_i \mathrm{mod} C_b) - 1}{C_b} + \left\lceil \frac{C_i}{C_b} \right\rceil \times (1 - \frac{(C_i \mathrm{mod} C_b) - 1}{C_b}) + \delta(C_i \mathrm{mod} C_b)$$

which is written after some manipulation as:

$$E(X) = \frac{C_i}{C_b} + \left\lceil \frac{C_i}{C_b} \right\rceil - \left\lfloor \frac{C_i}{C_b} \right\rfloor - \frac{1}{C_b} + \delta(C_i \mathrm{mod} C_b)$$

Where $\delta(k) = 1$ for $k=0$, 0 for $k \neq 0$.

References

Ahuja, S.R., C.S. Roberts, "An Associative/Parallel Processor for Partial Match Retrieval Using Superimposed Codes," Proc. 7th Annual Symp. on Computer Architecture, August 1980, pp.218-227.

Berra, P.B., E. Oliver, "The Role of Associative Array Processors in Data Base Machine Architecture," IEEE Computer, March 1979, pp.53-61.

Cardenas, A.F., "Analysis and Performance of Inverted Data Base Structures," Communications of the ACM Vol. 18, No. 5, May 1975, pp.253-263.

Clocksin, W.F. and Mellish, C.S., Programming in Prolog, Springer-Verlag, 1984

Colomb, R.M., Jayasooriah," A Clause Indexing System for Prolog based on Superimposed Coding," The Austrialian Computer Journal, Vol. 18, No. 1, February 1986, pp.18-25.

Fagin, R., J. Nievergelt, N. Pippenger and H.R. Strong, "Extendible Hashing-A Fast Access Method for Dynamic Files," ACM Transactions on Database Systems, Vol. 4, No. 3, September 1979, pp.315-344.

Faloutsos, C., "Access Methods for Text," Computing Surveys, Vol.17, No. 1, March 1985, pp.49-74.

Garg, A.K., C.C. Gotlieb, "Order-Preserving Key Transformations," ACM Transactions on Database Systems, Vol. 11, No. 2, June 1986, pp.213-234.

Kriddle, B. and McKusick, M.K., "Performance Effects of Disk Subsystem Choices for VAX systems Running 4.2 BSD UNIX," U.C. Berkeley C.S. Dep't Technical report , 1983.

Lee, D.L., "A Word-Parallel, Bit-Serial Signature Processor for Superimposed Coding," International Conference on Data Engineering, IEEE-CS, February 1986, pp.352-359.

Lloyd, J.W., "Optimal Partial-Match Retrieval," BIT 20, 1980, pp.406-413.

Lloyd, J.W., and K. Ramamohanarao, "Partial-Match Retrieval for Dynamic Files," BIT 22, 1982, pp.150-168.

Roberts, C.S., "Partial Match Retrieval via the Method of Superimposed Codes," Proceedings of the IEEE, Vol. 67, No. 12, December 1979, pp.1624-1642.

Wise, M.J. and D. Powers, "Indexing Prolog Clauses via Superimposed Code Words and Field Encoded Words," International Symp. on Logic Programming, February 1984, pp.203-210.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.